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 Ukranian Government infrastructure was attacked by a wiper malware. The Ukranian 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:

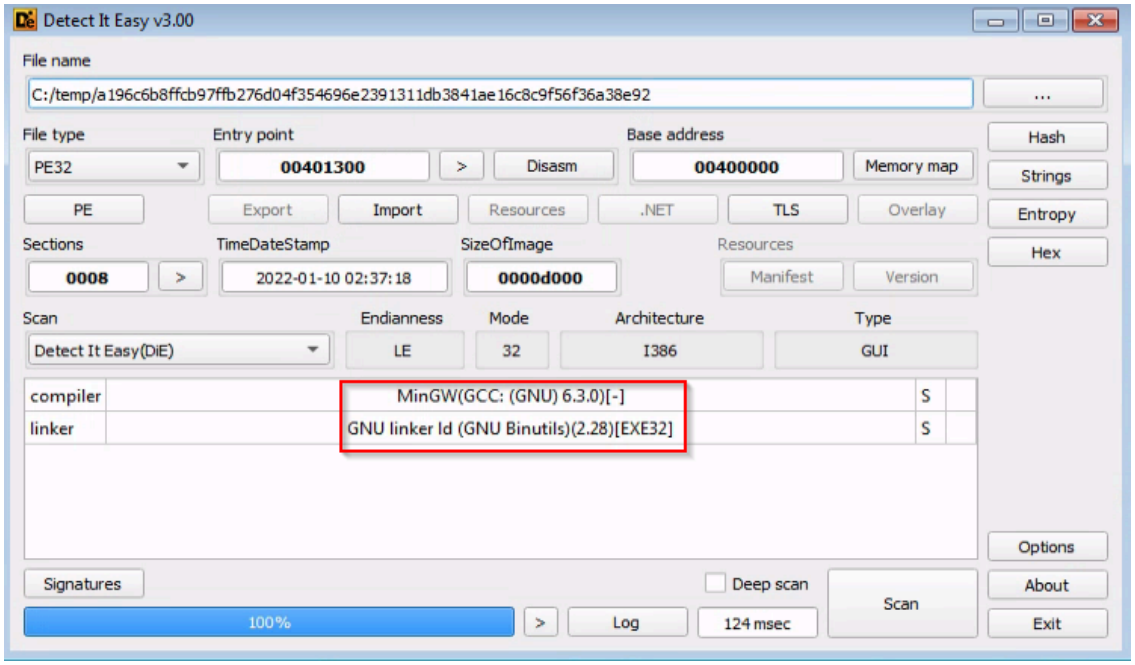

Figure 1: Stage1 compiler information

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.

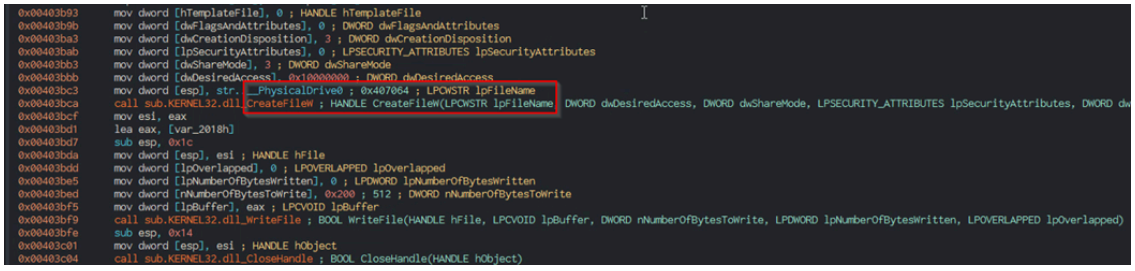

Figure 2: Code snippet responsible to MBR overwrite

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

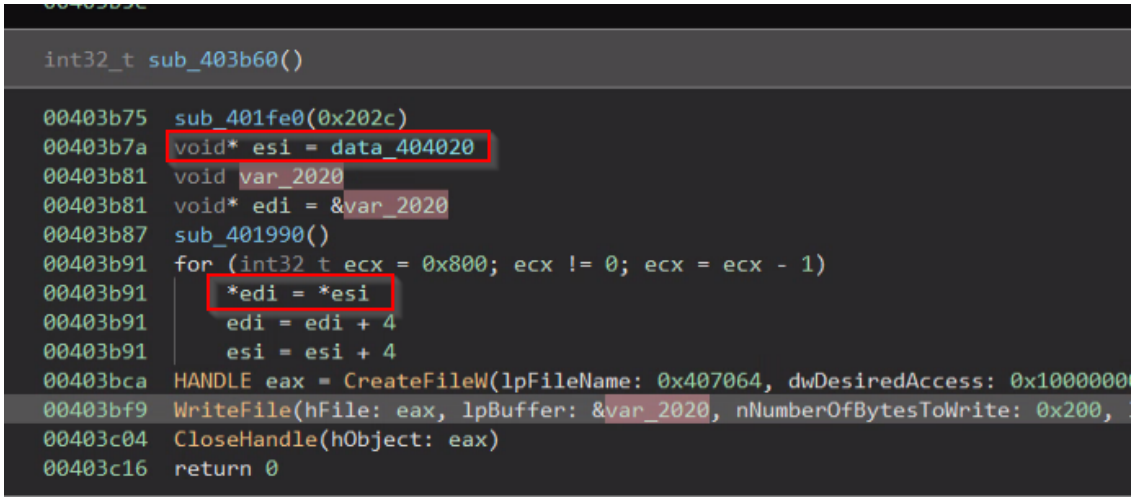

Figure 3: Highlighted code showing what will be written to the MBR

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

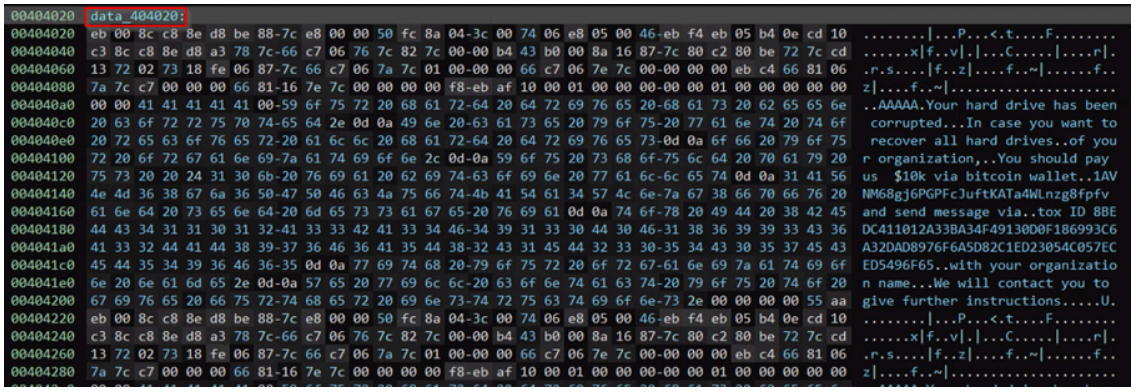

Figure 4: The actual data that will be written to the MBR

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.

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.

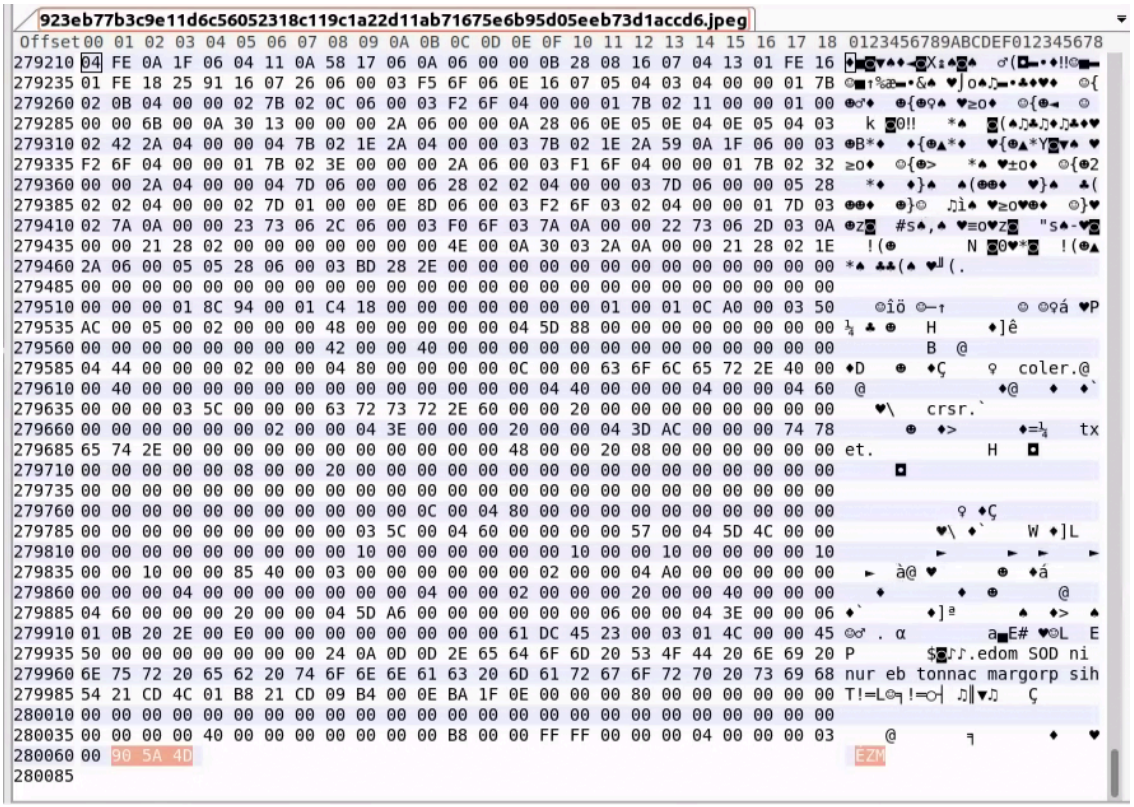

Figure 6: End of the downloaded file in hex editor

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:

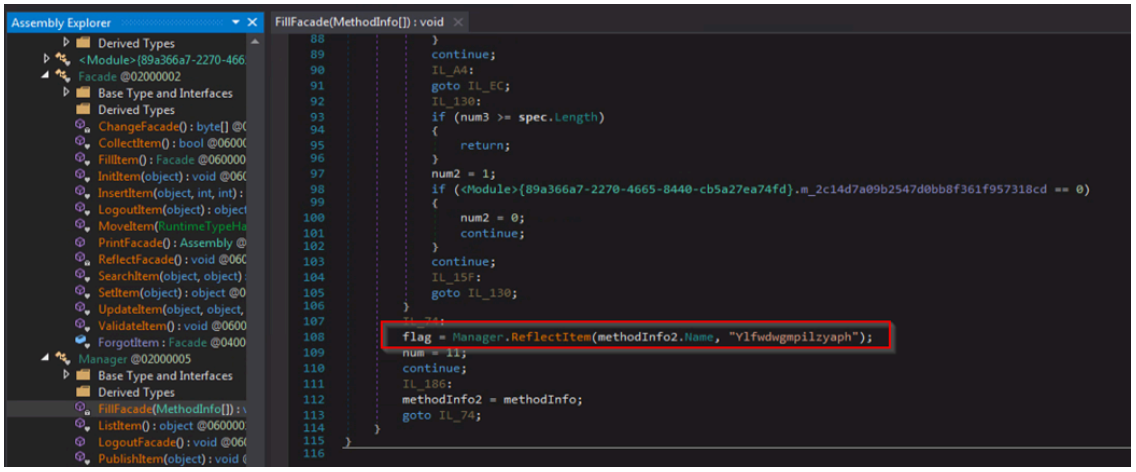

Figure 8: Calling method “Ylfwdwgmpilzyaph” from stage3

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 *“Ylfwdwgmpilzyaph” from this DLL*.

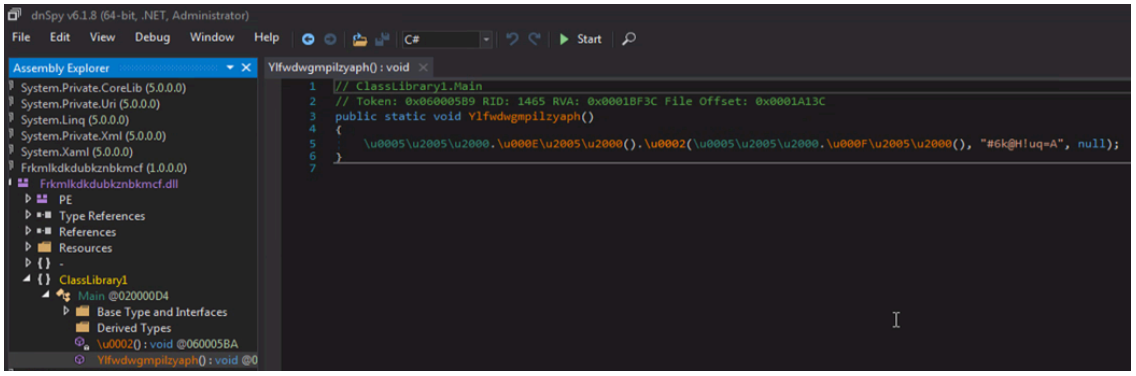

Figure 9: “Ylfwdwgmpilzyaph” code

The file contains three resources.

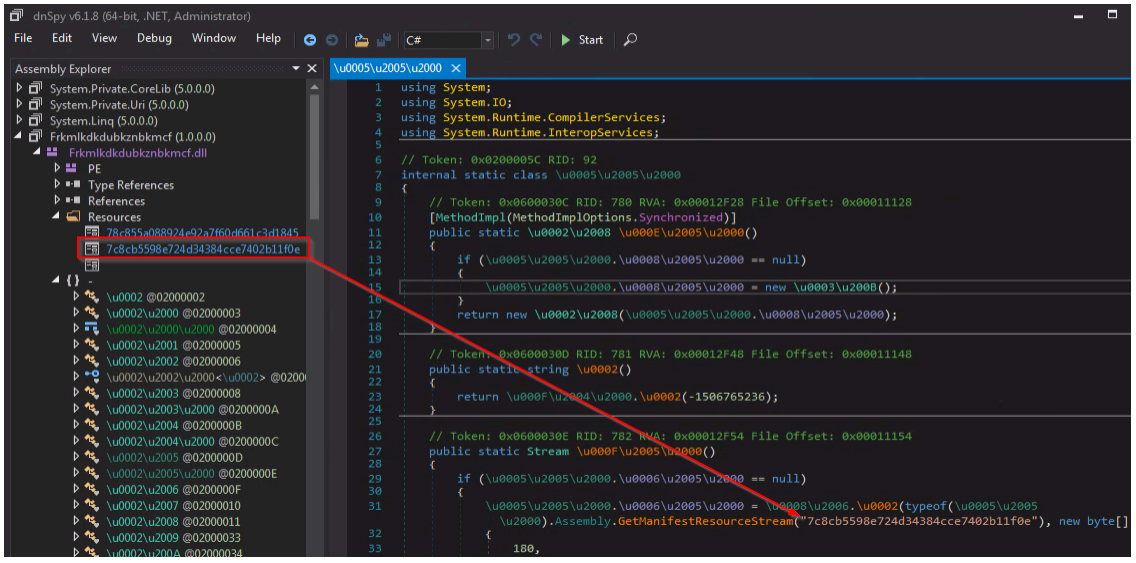

Figure 10: Code that loads resource “7c8cb5598e724d34384cce7402b11f0e”

Using "EazFixer" we were able to receive the decoded DLL file (35feefe6bd2b982cb1a5d4c1d094e8665c51752d0a6f7e3cae546d770c280f3a)

The decoded DLL contains two resources.

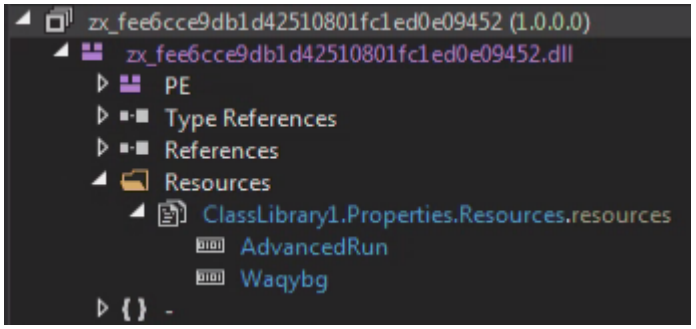

Figure 11: Resources in decoded DLL

The "**AdvancedRun**" resource is a gzip archive containing "AdvancedRun.exe," a legitimate application which is used by system administrators.

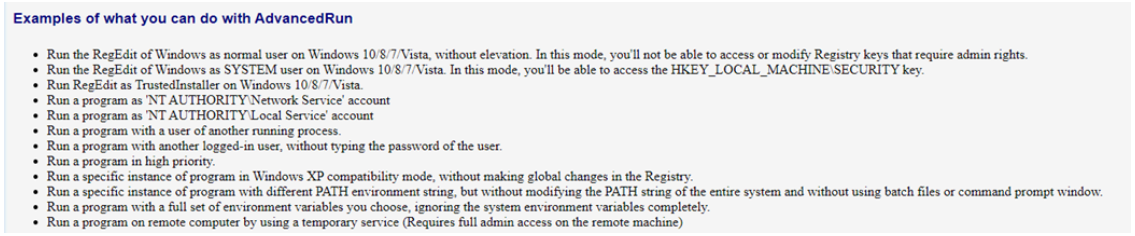

Figure 12: Official AdvancedRun usage examples

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:

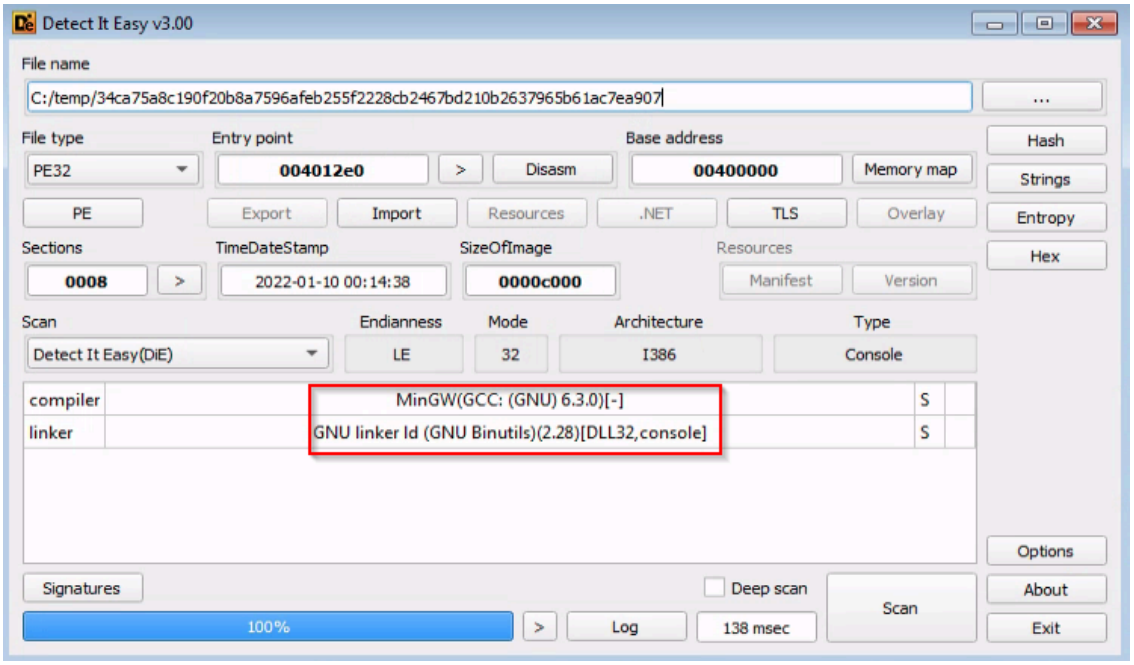

Figure 13: Stage4 compiler information

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

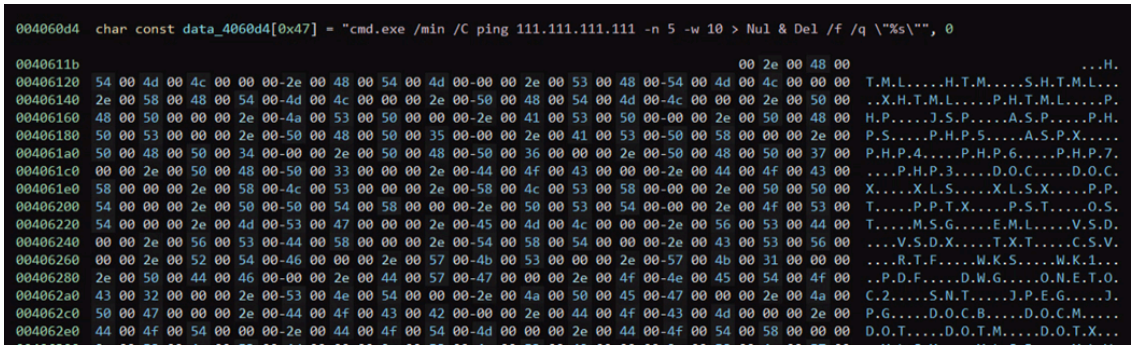

Figure 14: Delete command and partial extension list

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):

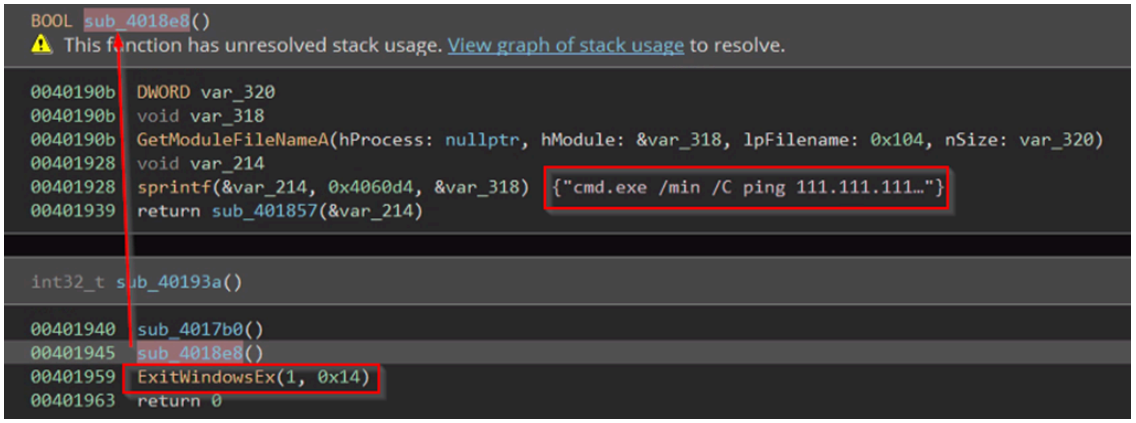

Figure 15: Exit routine

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:

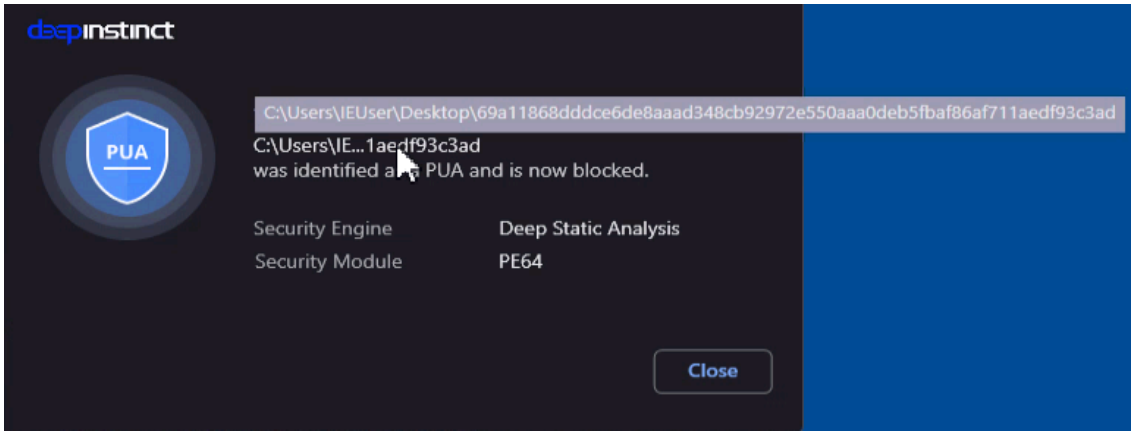

Figure 16: wmiexec being blocked

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

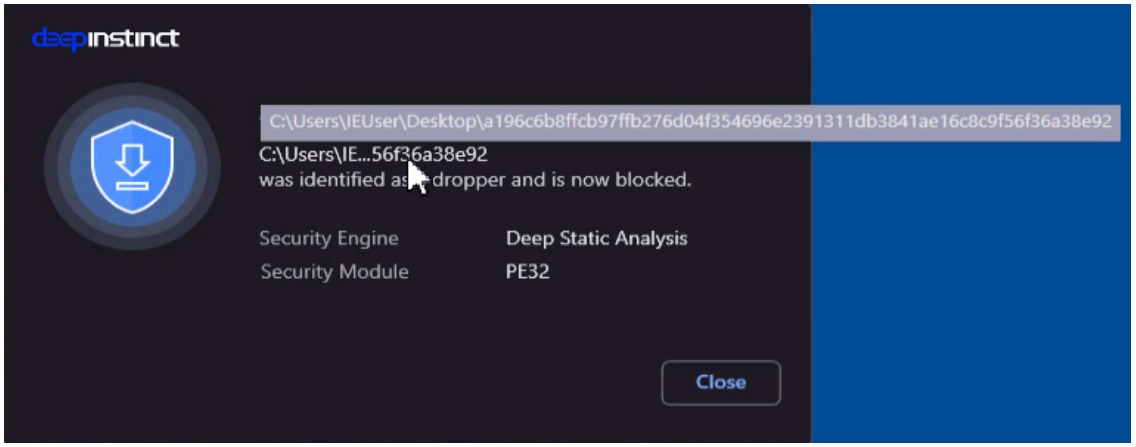

Figure 17: stage1 is being blocked

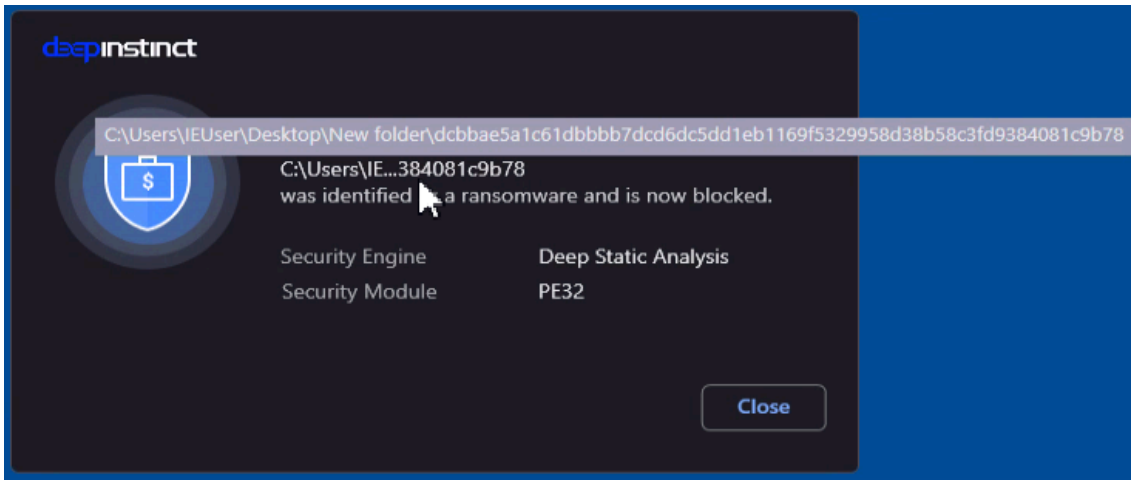

Figure 18: stage2 is being blocked

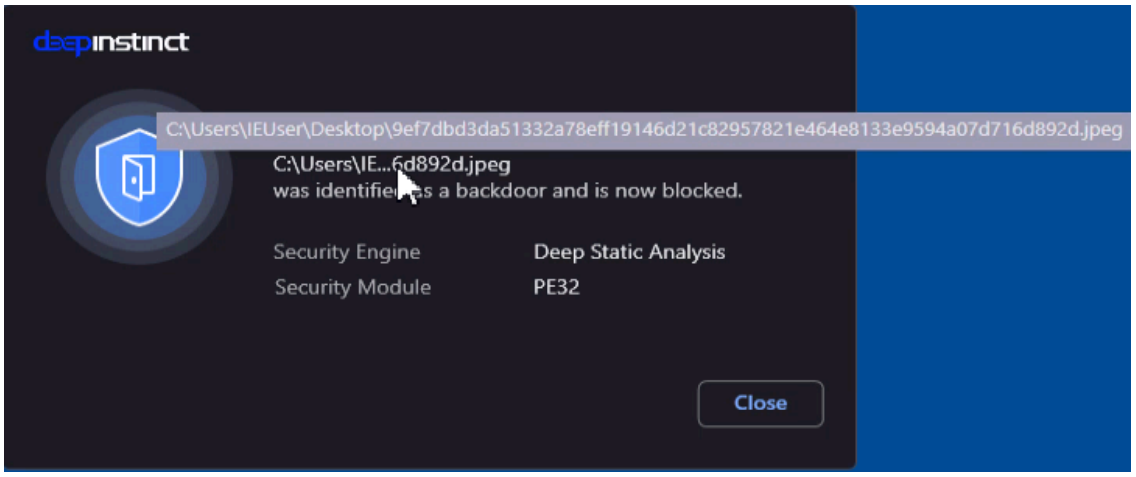

Figure 19: stage3 is being blocked

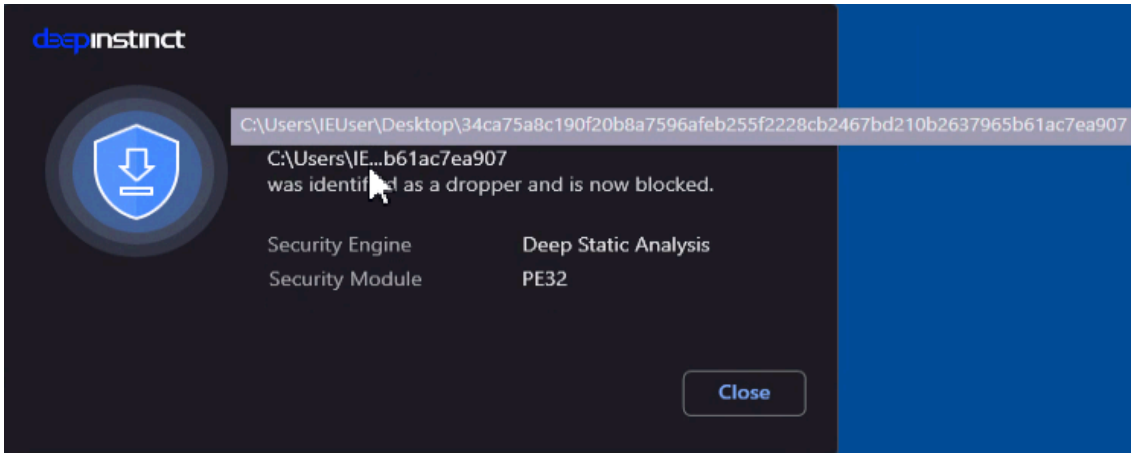

Figure 20: stage4 is being blocked

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