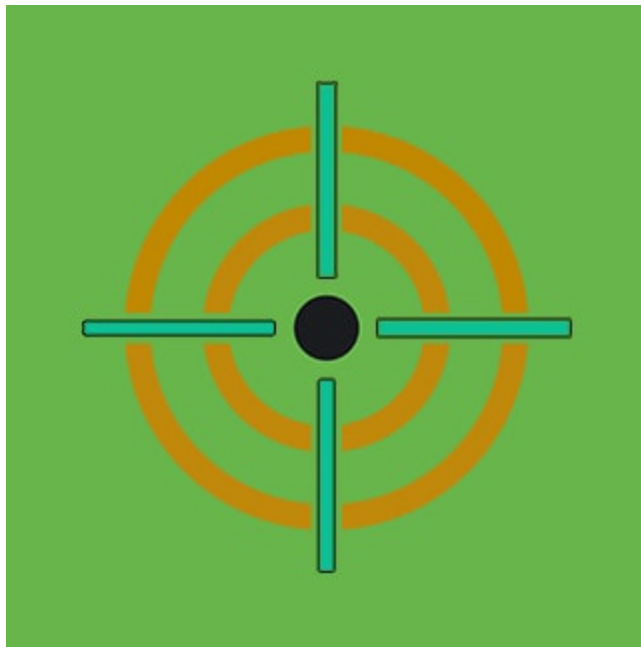


Threat Update DoubleZero Destructor

 splunk.com/en_us/blog/security/threat-update-doublezero-destroyer.html

March 28, 2022



By Splunk Threat Research Team March 28,

2022

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.

```

1 // ns0.GClass4
2 // Token: 0x0000004A RID: 74 RVA: 0x000442C File Offset: 0x0004262C
3 // Note: this type is marked as 'beforefieldinit'.
4 static GClass4()
5 {
6     GClass4.string_17 = Path.Combine(GClass4.smetho_0(), "Windows");
7     GClass4.string_18 = Path.Combine(GClass4.smetho_0(), "Windows", "Microsoft.NET");
8     GClass4.ienumerable_0 = new List<Regex>
9     (
10      {
11          new Regex(GClass4.smetho_0() + "\\Users\\.*?\\Local Settings.*", RegexOptions.IgnoreCase | RegexOptions.Compiled),
12          new Regex(GClass4.smetho_0() + "\\Users\\.*?\\AppData\\Local\\Application Data.*", RegexOptions.IgnoreCase |
13              RegexOptions.Compiled),
14          new Regex(GClass4.smetho_0() + "\\Users\\.*?\\Start Menu.*", RegexOptions.IgnoreCase | RegexOptions.Compiled),
15          new Regex(GClass4.smetho_0() + "\\Users\\.*?\\Application Data.*", RegexOptions.IgnoreCase |
16              RegexOptions.Compiled),
17          new Regex(GClass4.smetho_0() + "\\ProgramData\\Microsoft.*", RegexOptions.IgnoreCase | RegexOptions.Compiled)
18      });
19     GClass4.ienumerable_1 = new List<Regex>();
20     GClass4.ienumerable_2 = new List<Regex>
21     (
22      {
23          new Regex(GClass4.smetho_0() + "\\Users\\.*?\\AppData\\Local\\Microsoft.*", RegexOptions.IgnoreCase |
24              RegexOptions.Compiled),
25          new Regex(GClass4.smetho_0() + "\\Users\\.*?\\AppData\\Roaming\\Microsoft.*", RegexOptions.IgnoreCase |
26              RegexOptions.Compiled)
27      });
28     GClass4.ienumerable_3 = new List<Regex>();
29     GClass4.ienumerable_4 = new List<string>
30     (
31      {
32          Path.Combine(new string[]
33          {
34              Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles)
35          });
36          Path.Combine(new string[]
37          {
38              Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86)
39          });
40          Path.Combine(GClass4.smetho_0(), "Documents and Settings"),
41          Path.Combine(GClass4.smetho_0(), "ProgramData", "Application Data"),
42          Path.Combine(GClass4.smetho_0(), "Users", "All Users"),
43          Path.Combine(GClass4.smetho_0(), "Users", "Default User")
44      });
45     GClass4.ienumerable_5 = new List<string>
46     (
47      {
48          Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.System), "drivers")
49      });
50     GClass4.ienumerable_6 = new List<string>
51     (
52      {
53          Path.Combine(GClass4.smetho_0(), "Windows", "NTDS")
54      });
55     GClass4.ienumerable_7 = new List<string>
56     (
57      {
58          Path.Combine(GClass4.smetho_0(), "Windows")
59      });
60     };
61 }

```

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.


```

10,
180
});
while (l < 13)
{
    int num2 = jjs9HVgyl2cFR0.Op3pRI0B28vZwOn((int[])array4[1], 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
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
}
int num2 = jjs9HVgyl2cFR0.Op3pRI0B28vZwOn((int[])array2[j], 0, 0);
if (num2 <= 193)
{
    if (num2 != 55)
    {
        if (num2 != 177)
        {
            if (num2 == 193)
            {
                GClass6.RtlAdjustPrivilege(9UL, true, false, ref flag);
            }
            else
            {
                GClass6.RtlAdjustPrivilege(18UL, true, false, ref flag);
            }
        }
    }
    else if (num2 != 242)
    {
        if (num2 != 297)
        {
            if (num2 != 322)
            {
            }
        }
        else
        {
            GClass6.RtlAdjustPrivilege(17UL, true, false, ref flag);
        }
    }
    else
    {
        GClass6.RtlAdjustPrivilege(19UL, true, false, ref flag);
    }
}
j++;
}
catch (Exception)
{
}
}

```

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
00 %
method_2(string): bool
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
}
new int[]
{
    90,
    -1695743555,
    90,
    189035017,
    10,
    90,
    1506708663,
    10,
    180
}
};
while (m < 5)
{
    int num3 = aG1ItH3wd0Hc.JwbyZ0j7ANqRl((int[])array5[m], 0, 0);
    if (num3 != 79)
    {
        if (num3 != 124)
        {
        }
        else
        {
            num2 = GClass6.NtFsControlFile(safeFileHandle, IntPtr.Zero, IntPtr.Zero, IntPtr.Zero, ref gstruct2, 622792UL, IntPtr.Zero, (ulong)((long)Marshal.SizeOf(gstruct4)),
            IntPtr.Zero, (ulong)((long)aG1ItH3wd0Hc.JwbyZ0j7ANqRl(new int[]
            {
                90,
                1162427705,
                90,
                1162427705,
                100,
                210,
                200
            }
            ));
        }
    }
}
}

```

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**.

The screenshot displays a Windows Task Manager window with the 'Details' tab selected. The process list shows multiple instances of 'doublezero.exe' running. The 'Command Name' column is highlighted, showing various system control and file management actions such as 'Control: FSCTL_SET_ZERO_DATA', 'Control: FSCTL_SET_ZERO_DATA', and 'Control: FSCTL_SET_ZERO_DATA'. A red circle highlights the 'MimeWriter.py' file in the 'Command Name' column. A yellow circle highlights the 'MimeWriter.py' file in the 'File Name' column of the File Explorer window. The File Explorer window shows the content of 'MimeWriter.py', which is a Python module defining the 'MimeWriter' class. A red box labeled 'BEFORE' is placed over the code in the File Explorer window.

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.JwbyZ0j7ANqJR1(new int[]
        {
            90,
            1162427705,
            90,
            1162427705,
            190,
            210,
            220,
            190,
            210,
            220,
            270,
            230,
            180
        }, 0, 0), (int)num6);
    }
    else
    {
        fileStream.Write(array2, aG1ItHZ3wd0hGc.JwbyZ0j7ANqJR1(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),
        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.

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`
```

New Search Save As ▾ Creat

```

| 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

```

✓ 47 events (before 28/03/2022 12:18:56.000) No Event Sampling ▾ Job ▾ || ☰ ↻ 🗑

Events Patterns **Statistics (8)** Visualization

20 Per Page ▾ Format Preview ▾

s_name	process_path	process	proc_guid	registry_path	registry_value_name	registry_value_data	registry_key_name	action
zero_s.exe	C:\Temp\doublezero_s.exe	"C:\Temp\doublezero_s.exe"	{9531C931-5194-623C-9085-000000004302}	HKLM\System\CurrentControlSet\Control\Class\{4d36e96c-e325-11c0-bf01-000000004302}\ConfigurationVariables\FriendlyName	unknown	unknown	HKLM\System\CurrentControlSet\Control\Class\{4d36e96c-e325-11c0-bf01-000000004302}\ConfigurationVariables\FriendlyName	deleted
zero_s.exe	C:\Temp\doublezero_s.exe	"C:\Temp\doublezero_s.exe"	{9531C931-5194-623C-9085-000000004302}	HKLM\System\CurrentControlSet\Control\Class\{4d36e96c-e325-11c0-bf01-000000004302}\ConfigurationVariables\FriendlyName	unknown	unknown	HKLM\System\CurrentControlSet\Control\Class\{4d36e96c-e325-11c0-bf01-000000004302}\ConfigurationVariables\FriendlyName	deleted
zero_s.exe	C:\Temp\doublezero_s.exe	"C:\Temp\doublezero_s.exe"	{9531C931-5194-623C-9085-000000004302}	HKLM\System\CurrentControlSet\Control\Class\{6bdd1fc6-810f-11d0-bec7-000000004302}\ConfigurationVariables\FriendlyName	unknown	unknown	HKLM\System\CurrentControlSet\Control\Class\{6bdd1fc6-810f-11d0-bec7-000000004302}\ConfigurationVariables\FriendlyName	deleted
zero_s.exe	C:\Temp\doublezero_s.exe	"C:\Temp\doublezero_s.exe"	{9531C931-5194-623C-9085-000000004302}	HKLM\System\CurrentControlSet\Control\Class\{6d8b7884-7d21-11c0-bf01-000000004302}\ConfigurationVariables\FriendlyName	unknown	unknown	HKLM\System\CurrentControlSet\Control\Class\{6d8b7884-7d21-11c0-bf01-000000004302}\ConfigurationVariables\FriendlyName	deleted
zero_s.exe	C:\Temp\doublezero_s.exe	"C:\Temp\doublezero_s.exe"	{9531C931-5194-623C-9085-000000004302}	HKU\DEFAULT\Software\Microsoft\Windows\CurrentVersion\Run	unknown	unknown	HKU\DEFAULT\Software\Microsoft\Windows\CurrentVersion\Run	deleted
zero_s.exe	C:\Temp\doublezero_s.exe	"C:\Temp\doublezero_s.exe"	{9531C931-5194-623C-9085-000000004302}	HKU\DEFAULT\Software\Microsoft\Windows\CurrentVersion\Shell Extensions	unknown	unknown	HKU\DEFAULT\Software\Microsoft\Windows\CurrentVersion\Shell Extensions	deleted

Name	Technique ID	Tactic	Description
<u>Executables Or Script Creation In Suspicious Path</u>	<u>T1036</u>	Defense Evasion	This analytic will identify suspicious executable or scripts (known file extensions) in a list of suspicious file paths in Windows.
<u>Suspicious Process File Path</u>	<u>T1543</u>	Persistence, Privilege Escalation	This analytic will detect a suspicious process running in a file path where a process is not commonly seen and is most commonly used by malicious software.
<u>Windows Terminating Lsass Process (New)</u>	<u>T1562.001</u>	Defense Evasion	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 a security policy. This technique was seen in double zero malware that tries to wipe files and registry in compromised hosts.

<u>Windows Deleted Registry By A Non Critical Process File Path (New)</u>	<u>T1112</u>	Defense Evasion	This analytic is to detect deletion of registry with suspicious process file path.
---	--------------	-----------------	--

Filename - Sha256 description

Double Zero malware	3b2e708eaa4744c76a633391cf2c983f4a098b46436525619e5ea44e105355fe
---------------------	--

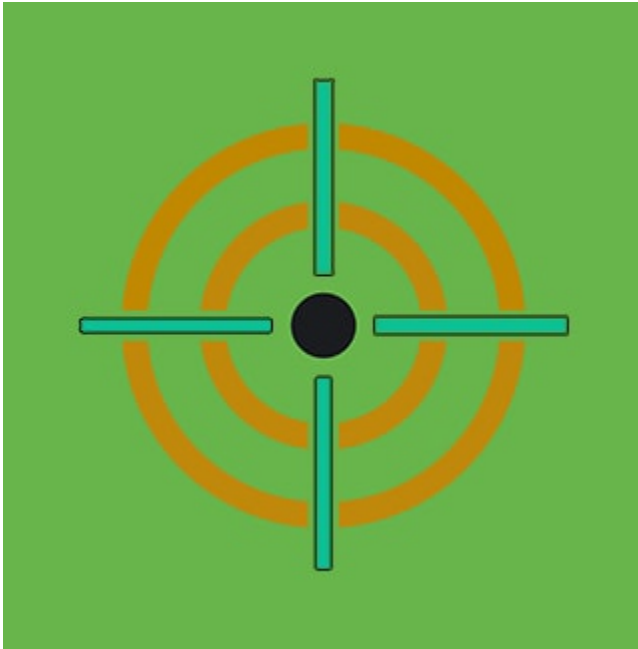
Learn More

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



Posted by

Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the [Attack Data repository](#).

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more [Splunk Security Content](#).