MALWARE/
TOOLS
PROFILE

Recorded Future®

By Insikt Group®

March 24, 2022

IsaacWiper Continues Trend of Wiper Attacks Against Ukraine

This report is a technical overview of the IsaacWiper malware reported by ESET on March 1, 2022. The malware was primarily delivered to Ukrainian organizations coincident with the Russian invasion of Ukraine. It is intended for those looking for a high-level overview of the malware's tactics, techniques, and procedures (TTPs) and mitigations.

Executive Summary

Following recent wiper attacks against Ukrainian organizations involving the WhisperGate and HermeticWiper malware, a new destructive wiper, IsaacWiper, was observed on February 24, 2022. Although no direct attribution for IsaacWiper or the other wiper malware found targeting Ukraine has been made by researchers, the timing of these destructive attacks in conjunction with tensions and kinetic conflict in Ukraine suggests they are Russian in origin.

Key Findings

- IsaacWiper is a destructive malware that overwrites all physical disks and logical volumes on a victim's machine.
- There is no code overlap between IsaacWiper, HermeticWiper, or WhisperGate. IsaacWiper achieves a similar outcome by different means.

Background

In a report released on March 1, 2022, ESET researchers identified a new destructive malware that had been affecting a Ukrainian government network since February 24, 2022. This malware, dubbed IsaacWiper, is distinct from the HermeticWiper malware previously reported. ESET stated that they observed IsaacWiper deployed at a Ukrainian organization that was not previously affected by HermeticWiper, although they are still assessing any links between IsaacWiper and HermeticWiper. Furthermore, they are unable to attribute this malware to any known threat actors due to a lack of significant code similarities with known malware. One day after IsaacWiper was initially deployed, the threat actors deployed a second version that included debug log output, indicating that the malware was not working as intended.

Technical Analysis

It is not known what method was used for initial access to the Ukrainian government network that IsaacWiper was discovered on. ESET researchers suggested it is likely that a tool such as Impacket was used for lateral movement within the network and that RemCom RAT was deployed on several systems at the same time as IsaacWiper.

IsaacWiper

IsaacWiper is a data destruction malware compiled with Visual Studio 2015 and written in a combination of C, C++, and assembly languages. ESET has observed it being deployed as both an EXE and a DLL from the *%programdata%* or *c:\windows\System32* directories with at least 5 filenames: *clean.exe, cl.exe, cl64.dll, cld.dll,* and *cll.dll*. For DLL samples, the original filename is *Cleaner.dll*. ESET noted that the earliest known PE compilation timestamp for IsaacWiper is from October 19, 2021.

Recorded Future analyzed a DLL sample of IsaacWiper with debug log capabilities. An example of the debug log output from a system with multiple physical and logical drives connected is provided in FIgure 1 below.

```
getting drives...

physical drives:
-- system physical drive 0: PhysicalDrive0
-- physical drive 1: PhysicalDrive1
-- physical drive 2: PhysicalDrive2

logical drives:
-- system logical drive: C:
-- logical drive: D:
-- logical drive: E:
-- logical drive: F:

start erasing physical drives...

physical drive 2-- FAILED
physical drive 1-- FAILED
-- start erasing logical drive F:d
start erasing system physical drive...

system physical drive -- FAILED
start erasing system logical drive C:
```

Figure 1: Example contents of IsaacWiper's log file (Source: Recorded Future)

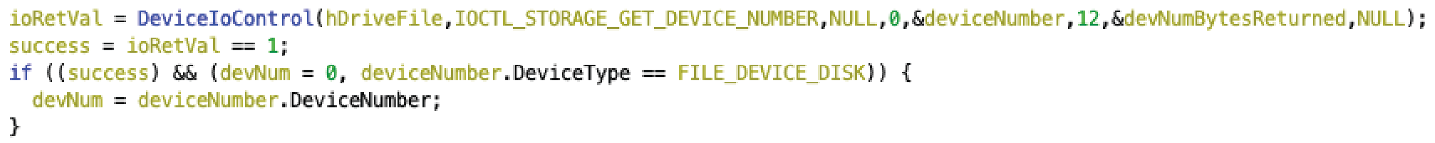
The DLL samples contain a single export function called _Start@4 that would typically be launched using a command similar to rundll32 <DLL_File>, #1. The wiper starts by creating a log file for its debug messages at %programdata%\log.txt. It then enumerates all physical disks on the system by creating a handle to \\PhysicalDrive<N>, where N starts at 0 and continues until an invalid handle is returned (indicating that the physical drive does not exist). Next, it determines the operating system's system directory by calling the GetWindowsDirectoryW API function. The drive letter for the system directory is extracted and then used to create a handle for \\.\<DriveLetter>:. An input/output control (IOCTL) call is then made to the drive's handle for IOCTL_STORAGE_DEVICE_NUMBER via DeviceIoControl, as shown in Figure 2 below.

This IOCTL call returns a STORAGE_DEVICE_NUMBER structure that provides the type, number, and partition number (if applicable) for a particular device. The device type is compared with the constant FILE_DEVICE_DISK to ensure that it is a device that contains partitions or can present itself as a disk before extracting the deviceNumber and storing it in a structure for later use.

Afterwards, the wiper makes another IOCTL call (IOCTL_DISK_GET_DRIVE_GEOMETRY_EX) to a handle for each physical drive to retrieve the size of the disk. The disk's size is stored in a structure representing the disk, as shown in Figure 3 below.

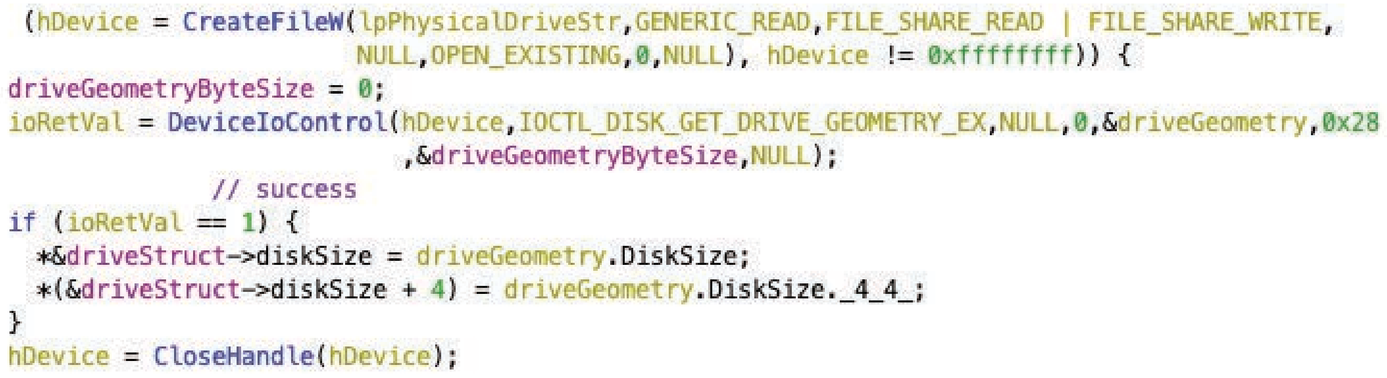
After enumerating the physical drives, the malware identifies the victim's logical drives with a call to GetLogicalDrives. For each logical drive, a handle is created for the file path pattern \\.\<DriveLetter>:, and then an IOCTL call is made with a control code of IOCTL_STORAGE_GET_DEVICE_NUMBER with each logical drive's handle.

After identifying all physical and logical drives, the wiper erases each of them. All physical drives (other than the one containing the OS) are wiped in a separate thread. Each thread makes an IOCTL call for control code FSCTL_LOCK_VOLUME to ensure the disk can only be used through a single handle. The wiper then generates random data using the Mersenne Twister pseudo-random number generator (PRNG) and subsequently writes it to the physical drive in 64 KB (0×10000 bytes) increments, as shown in Figure 4 below. This process continues until the *WriteFile* call fails, and then the volume is unlocked with an IOCTL call for control code FSCTL_UNLOCK_VOLUME and the thread terminates.



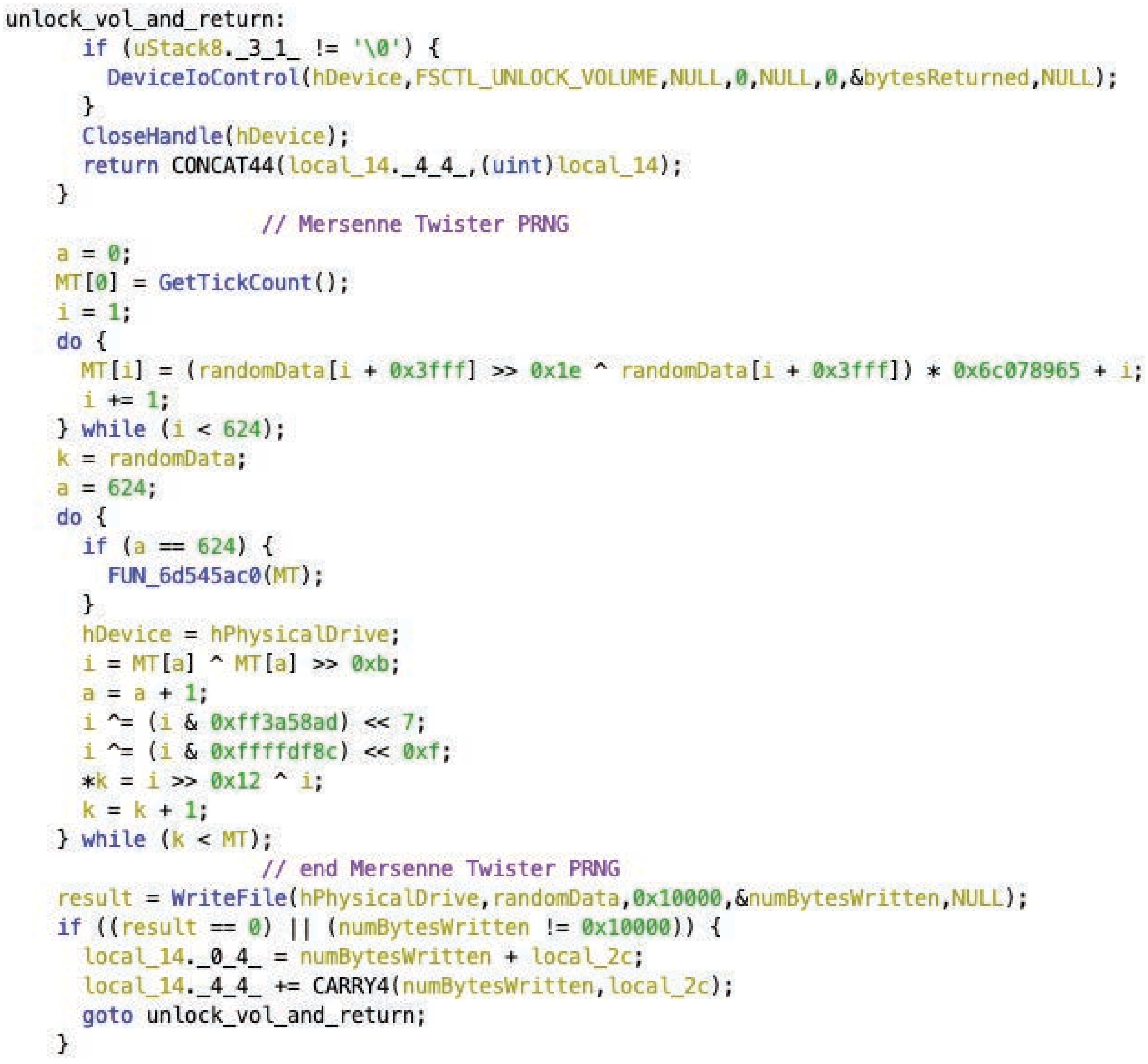
```
ioRetVal = DeviceIoControl(hDriveFile,IOCTL_STORAGE_GET_DEVICE_NUMBER,NULL,0,&deviceNumber,12,&devNumBytesReturned,NULL);
success = ioRetVal == 1;
if ((success) && (devNum = 0, deviceNumber.DeviceType == FILE_DEVICE_DISK)) {
  devNum = deviceNumber.DeviceNumber;
}
```

Figure 2: IOCTL call to retrieve a physical drive's type and device num ber (Source: Recorded Future)



```
  (hDevice = CreateFileW(lpPhysicalDriveStr,GENERIC_READ,FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,OPEN_EXISTING,0,NULL), hDevice != 0xffffffff)) {
 driveGeometryByteSize = 0;
 ioRetVal = DeviceIoControl(hDevice,IOCTL_DISK_GET_DRIVE_GEOMETRY_EX,NULL,0,&driveGeometry,0x28
                           ,&driveGeometryByteSize,NULL);
               // success
 if (ioRetVal == 1) {
   *&driveStruct->diskSize = driveGeometry.DiskSize;
   *(&driveStruct->diskSize + 4) = driveGeometry.DiskSize._4_4_;
 }
 hDevice = CloseHandle(hDevice);
```

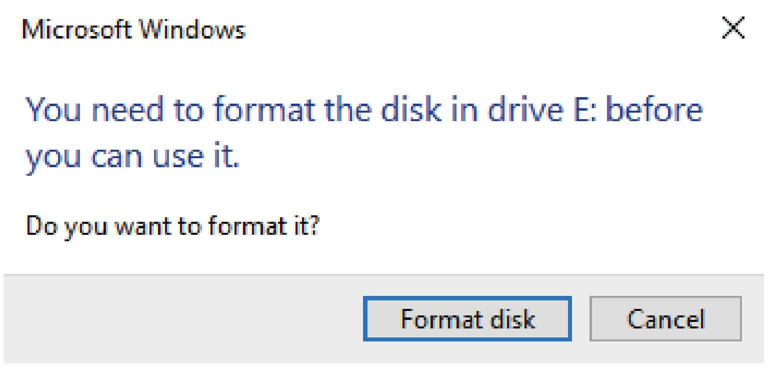
Figure 3: IOCTL to retrieve a physical drive's file size (Source: Recorded Future)



```
unlock_vol_and_return:
      if (uStack8._3_1_ != '\0') {
        DeviceIoControl(hDevice,FSCTL_UNLOCK_VOLUME,NULL,0,NULL,0,&bytesReturned,NULL);
      }
      CloseHandle(hDevice);
      return CONCAT44(local_14._4_4_,(uint)local_14);
    }
                    // Mersenne Twister PRNG
    a = 0;
    MT[0] = GetTickCount();
    i = 1;
    do {
      MT[i] = (randomData[i + 0x3fff] >> 0x1e ^ randomData[i + 0x3fff]) * 0x6c078965 + i;
      i += 1;
    } while (i < 624);
    k = randomData;
    a = 624;
    do {
      if (a == 624) {
        FUN_6d545ac0(MT);
      }
      hDevice = hPhysicalDrive;
      i = MT[a] ^ MT[a] >> 0xb;
      a = a + 1;
      i ^= (i & 0xff3a58ad) << 7;
      i ^= (i & 0xffffdf8c) << 0xf;
      *k = i >> 0x12 ^ i;
      k = k + 1;
    } while (k < MT);
                    // end Mersenne Twister PRNG
    result = WriteFile(hPhysicalDrive,randomData,0x10000,&numBytesWritten,NULL);
    if ((result == 0) || (numBytesWritten != 0x10000)) {
      local_14._0_4_ = numBytesWritten + local_2c;
      local_14._4_4_ += CARRY4(numBytesWritten,local_2c);
      goto unlock_vol_and_return;
    }
```

Figure 4: Physical drive overwrite with random data generated by the Mersenne Twister PRNG (Source: Recorded Future)

In our observations, the loop terminates with an error code 5 (“Access is denied”) on the last call to WriteFile. Subsequent attempts to access the file systems contained on those disks result in the user receiving a pop-up alert that it must be formatted, as shown in Figures 5 and 6.

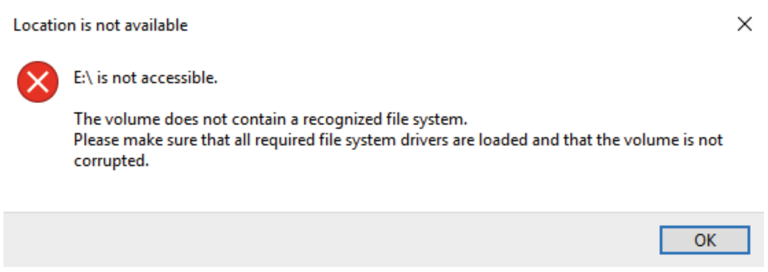


Microsoft Windows

You need to format the disk in drive E: before you can use it.

Do you want to format it?

Figure 5: Overwritten disk is unrecognized by Windows (Source: Recorded Future)



Location is not available

E:\ is not accessible.

The volume does not contain a recognized file system.
Please make sure that all required file system drivers are loaded and that the volume is not corrupted.

Figure 6: Error message when attempting to access an overwritten disk (Source: Recorded Future)

The wiper repeats this process for all the logical drives (with the exception of the logical drive containing the OS). A thread is created for each logical drive and it calls the same erase function used to overwrite the physical drives.

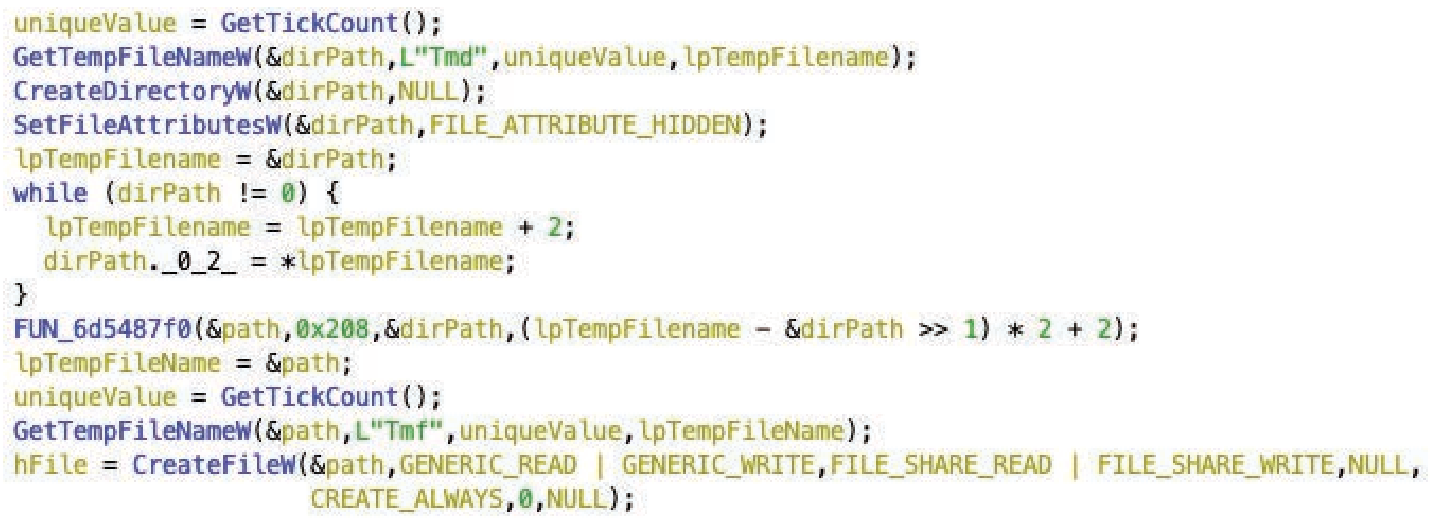
Once the logical drives are overwritten, the wiper moves on to the physical drive containing the OS. This process also leverages the same erasing function used in the previous 2 steps. The first 64 KB random chunks are written over the physical drive 16 times before the WriteFile API call fails with error code 5 ("Access is denied"). The Master Boot Record (MBR) of a physical drive is located in the first sector and is typically 512 KB, since the wiper's 16 overwrites start at the beginning of the disk and represent the first 1 MB of the physical drive they effectively overwrites the drive's MBR.

Finally, the wiper begins erasing the logical drive containing the OS. The same disk erase function used on all the other physical and logical drives is tried again; however, this time the IOCTL call for control code FSCTL_LOCK_VOLUME fails, and subsequent writes to the drive's file handle also result in an "Access is denied" error again. This is likely due to the physical drive already being overwritten until access was denied in the previous step.

Afterwards, the wiper spawns a new thread to fill the remaining free space on the OS's logical drive with random data. It creates a hidden directory under the C drive in the following format "Tmd<unique value>.tmp", and also creates a temporary filename in the hidden directory using a similar naming convention ("Tmf<unique value>.tmp"). Random data is then repeatedly generated with the Mersenne Twister PRNG algorithm and written to a temporary file until the disk runs out of space, as shown in Figure 7 below.

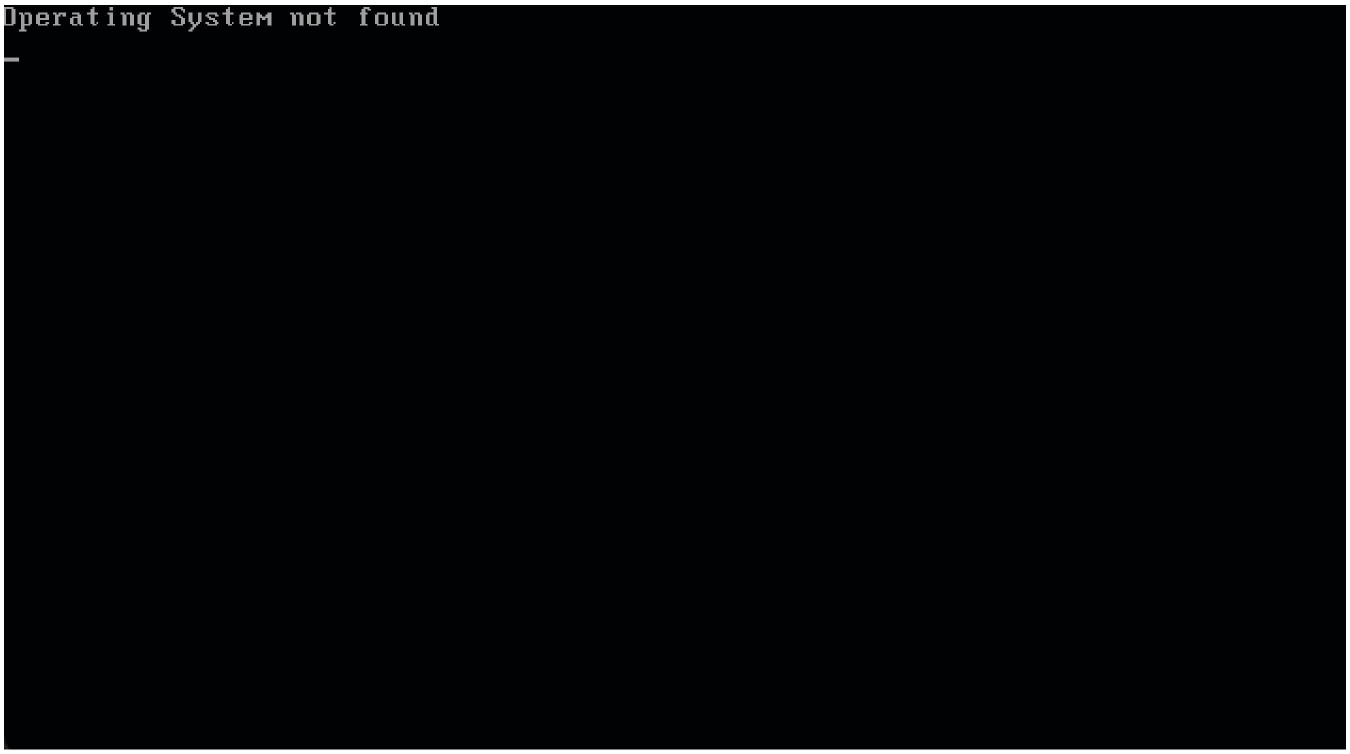
While that thread to fill the logical disk is running, the wiper's main thread recursively iterates through all files on the system logical drive. Each file is fully overwritten with random data in 64 KB increments.

Upon completion all drives and data on the system have been overwritten. When rebooting the system, a victim is presented with an error message similar to the one shown in Figure 8.



```
uniqueValue = GetTickCount();
GetTempFileNameW(&dirPath,L"Tmd",uniqueValue,lpTempFilename);
CreateDirectoryW(&dirPath,NULL);
SetFileAttributesW(&dirPath,FILE_ATTRIBUTE_HIDDEN);
lpTempFilename = &dirPath;
while (dirPath != 0) {
  lpTempFilename = lpTempFilename + 2;
  dirPath._0_2_ = *lpTempFilename;
}
FUN_6d5487f0(&path,0x208,&dirPath,(lpTempFilename - &dirPath >> 1) * 2 + 2);
lpTempFileName = &path;
uniqueValue = GetTickCount();
GetTempFileNameW(&path,L"Tmf",uniqueValue,lpTempFileName);
hFile = CreateFileW(&path,GENERIC_READ | GENERIC_WRITE,FILE_SHARE_READ | FILE_SHARE_WRITE,NULL,
                    CREATE_ALWAYS,0,NULL);
```

Figure 7: Code to generate a temporary file used to fill the disk with random data (Source: Recorded Future)



```
Operating System not found
_
```

Figure 8: Failed boot attempt after IsaacWiper infection (Source: Re corded Future)

Mitigations

Although there are no hard indicators of an IsaacWiper infection In log data, indications of the deployment of IsaacWiper and other wipers have certain generic indicators that defenders can monitor for.

On Windows systems, alerts such as “Suspicious remote activity”, “Suspicious access to LSASS service”, and “Microsoft Defender Antivirus tampering” can provide early indications of an attack using TTPs similar to those used deploying HermeticWiper and IsaacWiper.

Microsoft also suggests watching for phishing campaigns using Ukrainian-associated ruses. The following domains are examples cited by Microsoft used in wiper attacks on Ukrainian organizations:

- help-for-ukraine[.]eu
- tokenukraine[.]com
- ukrainesolidarity[.]org
- ukraine-solidarity[.]com
- saveukraine[.]today
- supportukraine[.]today

Wiper programs in general seek to destroy data within the target system or against a targeted device. Potential targets can mitigate the impact of a successful wiper attack by instituting regular backup policies. These attacks can be further mitigated by creating a disaster recovery plan in order to make backup data highly available and reduce downtime after an attack.

Outlook

Based on our analysis, IsaacWiper does not share code with either of the previous wipers observed attacking Ukraine (WhisperGate and HermeticWiper); however, attack objectives and targeting methodologies associated with IsaacWiper are similar to those observed with WhisperGate and HermeticWiper. While there is currently no definitive attribution related to the creators of IsaacWiper or the threat actors that have carried out attacks using IsaacWiper, this data further demonstrates that Ukranian entities are at increased risk of targeting in wiper-related attacks.

Appendix A: YARA Rules

```
import "pe"

rule MAL_IsaacWiper {
    meta:
        author = "CNANCE, Insikt Group, Recorded Future"
        date = "2022-03-08"
        description = "Detects IsaacWiper destructive malware"
        version = "1.0"
           reference = "https://www.welivesecurity.com/2022/03/01/isaacwiper-hermeticwizard-wiper-worm-targeting-
ukraine/"
        hash = "13037b749aa4b1eda538fda26d6ac41c8f7b1d02d83f47b0d187dd645154e033"
        hash = "7bcd4ec18fc4a56db30e0aaebd44e2988f98f7b5d8c14f6689f650b4f11e16c0"
        RF_MALWARE = "IsaacWiper"
        RF_MALWARE_ID = "lzQ5GL"

    strings:
            /*
                6d5473f8 6a 00          PUSH       0x0
                6d5473fa 6a 00          PUSH       0x0
                6d5473fc 6a 03          PUSH       0x3
                6d5473fe 6a 00          PUSH       0x0
                6d547400 6a 03          PUSH       0x3
                6d547402 68 00 00       PUSH       0x80000000
                         00 80
                6d547407 8d 45 cc       LEA        hCDrive=>str_\\\\.\\C:,[EBP + -0x34]
                6d54740a 50             PUSH       hCDrive
                6d54740b ff d7          CALL       EDI=>KERNEL32.DLL::CreateFileW
                6d54740d 8b f0          MOV        ESI,hCDrive
                6d54740f 83 fe ff       CMP        ESI,-0x1
                6d547412 0f 84 c6       JZ         LAB_6d5474de
                         00 00 00
                6d547418 6a 00          PUSH       0x0
                6d54741a 8d 45 e4       LEA        hCDrive=>bytesReturned,[EBP + -0x1c]
                6d54741d c7 45 e4       MOV        dword ptr [EBP + bytesReturned],0x0
                         00 00 00 00
                6d547424 50             PUSH       hCDrive
                6d547425 6a 0c          PUSH       12
                6d547427 8d 45 ac       LEA        hCDrive=>storageDeviceNumber,[EBP + -0x54]
                6d54742a 50             PUSH       hCDrive
                6d54742b 6a 00          PUSH       0x0
                6d54742d 6a 00          PUSH       0x0
                6d54742f 68 80 10       PUSH       IOCTL_STORAGE_GET_DEVICE_NUMBER
                         2d 00
                6d547434 56             PUSH       ESI
                6d547435 ff 15 00       CALL       dword ptr [->KERNEL32.DLL::DeviceIoControl]      = 00034520
                         60 56 6d
                6d54743b 83 f8 01       CMP        retVal,0x1
                6d54743e 0f 94 c3         SETZ        BL
                6d547441 75 0d            JNZ         LAB_6d547450
                6d547443 33 c0            XOR         retVal,retVal
                6d547445 83 7d ac 07      CMP         dword ptr [EBP + storageDeviceNumber.DeviceTyp
                6d547449 0f 44 45 b0      CMOVZ       retVal,dword ptr [EBP + storageDeviceNumber.De
                6d54744d 89 45 fc         MOV         dword ptr [EBP + deviceNumber],retVal
                                    LAB_6d547450                                    XREF[1]:     6d547441(j)
                6d547450 56               PUSH        ESI
               6d547451 ff 15 10         CALL       dword ptr [->KERNEL32.DLL::CloseHandle]          = 00034566
                         60 56 6d
                6d547457 84 db            TEST        BL,BL
                6d547459 eb 02            JMP         LAB_6d54745d
                                    LAB_6d54745b                                    XREF[1]:     6d54736c(j)
                6d54745b 84 c9            TEST        diskStructParam,diskStructParam
                                    LAB_6d54745d                                    XREF[1]:     6d547459(j)
                6d54745d 0f 84 7b         JZ          LAB_6d5474de
                         00 00 00
                6d547463 8b 5d fc         MOV         EBX,dword ptr [EBP + deviceNumber]
                6d547466 8b d3            MOV         otherDiskStructParam,EBX
                6d547468 8b 4d e0         MOV         diskStructParam,dword ptr [EBP + diskStructPar
                6d54746b 6a 01            PUSH        0x1
               6d54746d e8 de 9b         CALL       get_DiskGeometry                                undefined4
get_DiskGeometry(Disk
                         ff ff
        */
        $physical_drive_check = { 6a 00 6a 00 6a 03 6a 00 6a 03 68 00 00 00 80 8d ?? cc 50 ff d? 8b f0 83 fe
ff 0f 84 ?? ?? ?? ?? 6a 00 8d ?? e4 c7 4? ?? 00 00 00 00 50 6a 0c 8d ?? ac 50 6a 00 6a 00 68 80 10 2d 00 56
ff 15 ?? ?? ?? ?? 83 f8 01 0f 94 ?? 75 ?? 33 c0 83 7? ?? 07 0f 44 4? ?? 89 4? ?? 56 ff 15 ?? ?? ?? ?? 84 db
eb ?? 84 c9 0f 84 ?? ?? ?? ?? 8b 5? ?? 8b d3 8b 4? ?? 6a 01 e8 }

    condition:
        uint16(0) == 0x5a4d // PE file
        and filesize > 170KB
        and pe.imphash() == "a4b162717c197e11b76a4d9bc58ea25d"
        and pe.exports("_start@4")
        and pe.imports("kernel32.dll", "DeviceIoControl")
        and $physical_drive_check
}
```

Appendix B: IOCs

IsaacWiper Sample (SHA256):

```
13037b749aa4b1eda538fda26d6ac41c8f7b1d02d83f47b0d187dd645154e033
0c61e11f4b056f9866f41c8d5b7f89f8892e44dbeaa0e03bd65a4cf81ce4dcb7
```

About Insikt Group®

Insikt Group is Recorded Future's threat research division, comprising analysts and security researchers with deep government, law enforcement, military, and intelligence agency experience. Their mission is to produce intelligence on a range of cyber and geopolitical threats that reduces risk for clients, enables tangible outcomes, and prevents business disruption. Coverage areas include research on state-sponsored threat groups; financially-motivated threat actors on the darknet and criminal underground; newly emerging malware and attacker infrastructure; strategic geopolitics; and influence operations.

About Recorded Future

Recorded Future is the world's largest intelligence company. The Recorded Future Intelligence Platform provides the most complete coverage across adversaries, infrastructure, and targets. By combining persistent and pervasive automated data collection and analytics with human analysis, Recorded Future provides real-time visibility into the vast digital landscape and empowers clients to take proactive action to disrupt adversaries and keep their people, systems, and infrastructure safe. Headquartered in Boston with offices and employees around the world, Recorded Future works with more than 1,300 businesses and government organizations across 60 countries.

Learn more at recordedfuture.com and follow us on Twitter at @RecordedFuture.