



**REPUBLIC OF ALBANIA
NATIONAL AUTHORITY FOR CYBER SECURITY
DIRECTORATE OF CYBERSECURITY ANALYSIS**

**Technical Analysis of the Malicious File
Display_10.exe**

**Version: 1.0
Date: 25/06/2025**

TLP-CLEAR

Content

Technical Information.....	4
Analysis of the file Display_10.exe.....	4
Analysis of the rawio.sys file	11
Indicators of Compromise	14
Recommendations.....	14
Picture 1 Compiler Language and Certificate.....	4
Picture 2 Function FUN_1400294c0.....	5
Picture 3 The presence of rawio.sys in the Temp directory.....	5
Picture 4 Creation of the rawio.sys driver in runtime.....	6
Picture 5 file extensions.....	7
Figure 6 Searching of disks	8
Figure 7 Reading files and directories.....	8
Figure 8 Creation of the malicious service RAW_IO.....	9
Figure 9 sectorio.sys.....	10
Figure 10 Services of symantec ect.....	10
Figure 11 Certificate of rawio.sys	12
Figure 12 The RAW_IO.sys service	12
Picture 13 source code of sectorio.c	13
Picture 14 Post display_10.exe execution	13

This report has limitations and should be interpreted with caution.

Some of these limitations include:

Phase One:

Information Sources: This report is based on information available at the time of its preparation. Certain aspects may differ from ongoing or future developments.

Phase Two:

Analysis Details: Due to resource constraints, some aspects of the malicious file may not have been analyzed in depth. Any additional unknown information may result in changes to the findings presented in this report.

Phase Three:

Information Security: To protect sources and confidential data, certain details may be redacted or omitted. This decision was made to preserve the integrity and security of the information used during the investigation.

The Cybersecurity Analysis Directorate (AKSK) reserves the right to modify, update, or revise any part of this report without prior notice.

Disclaimer:

This report is not a final document.

The findings are based on information available at the time of investigation and analysis. There is no guarantee that the reported details will remain unchanged, and future updates may affect the current conclusions.

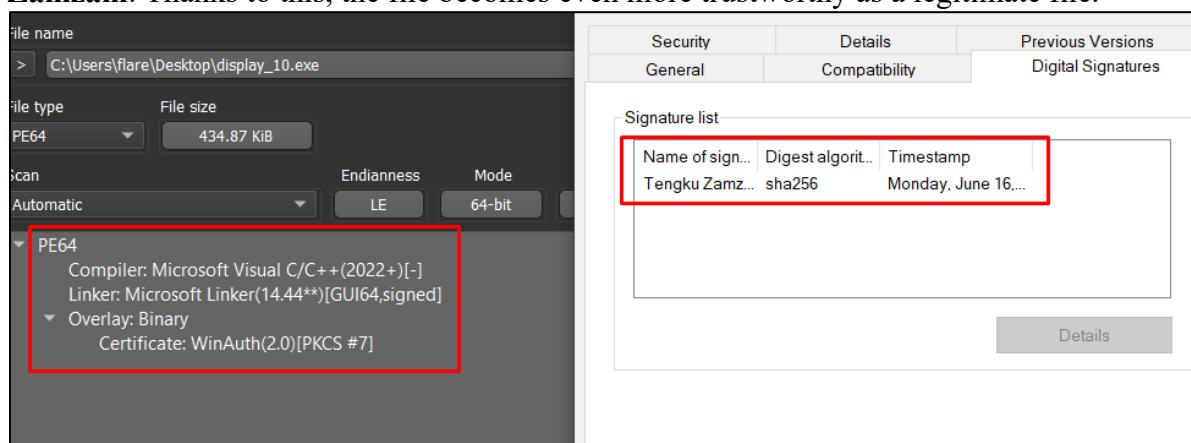
The authors of this report accept no responsibility for any misuse or for decisions made solely based on the contents of this report.

Technical Information

A malicious file has been detected in the IT systems of the Municipality of Tirana, where it was distributed and executed on the Windows server operating systems and endpoint devices. The file has a destructive role, and once executed, it damages the operating system, overwrites files such as .dll, .exe, .docx, etc., destroying their originality.

Analysis of the file Display_10.exe

The file **display_10.exe** is an executable file written in C/C++ language, which contains a legitimate certificate, meaning it is signed and officially verified under the name **Tengku Zamzam**. Thanks to this, the file becomes even more trustworthy as a legitimate file.



Picture 1 Compiler Language and Certificate

During the code analysis, several functionalities of this malicious file were identified.

FUN_1400294c0 that does the following actions:

- It checks the privileges of the current process (such as SeShutdownPrivilege).
- It creates a file on the disk in a specific location on the desktop.
- It prepares a list of files with names that appear to be DLL modules, EXE, sys, piz, gif, rar, z7, tar, etc.
- It processes these files with other internal functions such as **FUN_14002ac80**, **FUN_140027e58**, **FUN_14002a670**, **FUN_14002b554**.
- It performs a device/disk check, and the character string **C:\Windows\System32\drivers\beep.sys** has been seen.
- Finally, it attempts to gain the privilege to shut down or restart the system, and if successful, it calls **ExitWindowsEx..**

```

GetTokenInformation(local_640,TokenElevation,&local_5f8,4,(PDWORD)&local_5f0);
pauVar9 = local_640;
CloseHandle(local_640);
}
FUN_140025cc8(pauVar9,(undefined (*) [32])local_158);
DVar7 = GetFileAttributesA(local_158);
if ((DVar7 == 0xffffffff) && (DVar7 == GetLastError(), DVar7 == 2)) {
pvVar10 = CreateFileA(local_158,0x10000000,0,(LPSECURITY_ATTRIBUTES)0x0,2,0x80,(HANDLE)0x0);
if ((pvVar10 == (HANDLE)0xffffffff) ||
(WriteFile(pvVar10,&DAT_140065390,0x2d88,(LPDWORD)&local_5f8,(LPOVERLAPPED)0x0),
(int)local_5f8 != 0x2d88)) {
return 0xffffffff;
}
CloseHandle(pvVar10);
}
puVar11 = FUN_140035634(1);
plVar19 = (longlong *)0x18c;
pcVar18 = "C:\\Users\\Test\\Desktop\\currentproject\\unfreez\\scr\\main.cpp";
FUN_140025c84((ulonglong)puVar11,0x14005f318,
"C:\\Users\\Test\\Desktop\\currentproject\\unfreez\\scr\\main.cpp",0x18c);
pFVar12 = (FILE *)FUN_140035634(2);
fflush(pFVar12);
FUN_140025d40(local_158);
local_5a8 = 4;
uStack_5a4 = 0;
uStack_5a0 = 0xf;
uStack_59c = 0;
local_5b8, 5, 11 = SUB1611(ZEXT816(0),5);

```

Picture 2 Function FUN_1400294c0

The function FUN_140025cc8 retrieves the system's temporary file path (Temp) via GetTempPathA(0x104, local_118);

This Windows function retrieves the temporary path, for example:

C:\Users<username>\AppData\Local\Temp, and constructs a character string

C:\Users<username>\AppData\Local\Temp\rawio.sys.

This suggests that we are dealing with a driver file in an unusual directory.

```

1
2 undefined (*) [32] FUN_140025cc8(undefined8 param_1,undefined (*)param_2
3
4 {
5     CHAR local_118 [272];
6
7     FUN_14004bcc0(param_2,0,0x104);
8     GetTempPathA(0x104,local_118);
9     strncpy((char *)param_2,local_118,0x104);
10    strncat((char *)param_2,"\\",0x104);
11    strncat((char *)param_2,"rawio.sys",0x104);
12    return param_2;
13 }
14

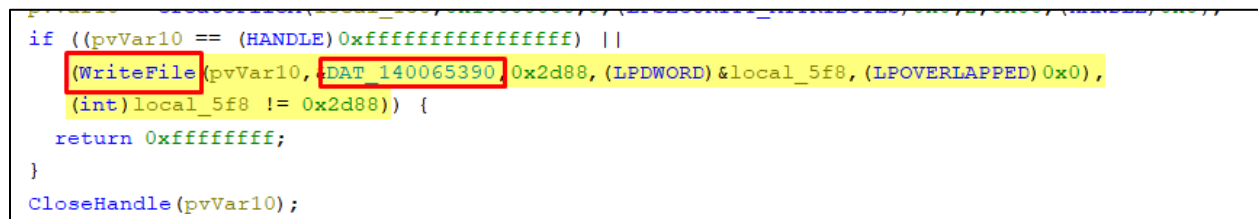
```

Picture 3 The presence of rawio.sys in the Temp directory.

If we go back to the previous function **FUN_1400294c0**, it is exactly here that the **rawio.sys** file is created..

```
pvVar10 = CreateFileA(local_158,
    0x10000000, // GENERIC_READ | GENERIC_WRITE
    0,
    NULL,
    2, // CREATE_ALWAYS
    0x80, // FILE_ATTRIBUTE_NORMAL
    NULL);
```

The **rawio.sys** file is created with write permission/read-write access. If it already exists, it is deleted and created from the beginning. In the code section below, it writes **0x2D88** bytes from the buffer **&DAT_140065390** to the file. This is the content of the **rawio.sys** driver being created on the disk.



```
if ((pvVar10 == (HANDLE)0xffffffffffffffff) ||
    (WriteFile(pvVar10, DAT_140065390, 0x2d88, (LPDWORD)&local_5f8, (LPOVERLAPPED)0x0),
    (int)local_5f8 != 0x2d88)) {
    return 0xffffffff;
}
CloseHandle(pvVar10);
```

Picture 4 Creation of the rawio.sys driver in runtime

In the following part of this function, the names and extensions are stored as hexadecimal values. Yes, this section of the code shows a group of file names with common extensions such as **.backup**, **.config**, **.gif**, **.pst**, **.ost**, **.bak**, **.exe**, **.dll**, etc. These are typically targets for ransomware or other malicious programs.

It then prepares to iterate over the structure containing these names.

This will later be used to:

This will later be used to:

- Scanning files on the disk
- Searching local disks for files that end with these extensions.

```
Decompile: FUN_1400294c0 - (display_10.exe)
...
_local_598 = ZEXT716(0x6769666e6f632e);
local_568 = 4;
uStack_564 = 0;
uStack_560 = 0xf;
uStack_55c = 0;
auStack_573 = SUB1611(ZEXT816(0),5);
local_578 = (undefined [5])0x7473702e;
local_548._8_4_ = 0xf;
local_548._0_8_ = 4;
uStack_53c = 0;
stack0xfffffffffffffaad = SUB1611(ZEXT816(0),5);
local_558._0_5_ = 0x74736f2e;
local_528 = 4;
uStack_524 = 0;
uStack_520 = 0xf;
uStack_51c = 0;
local_538._5_11_ = SUB1611(ZEXT816(0),5);
local_538._0_5_ = 0x6b61622e;
stack0xffffffffffff9d0 = (LUID)local_518;
_local_638 = (undefined (*) [32])local_5b8;
FUN_14002ac80((undefined (**) [16])local_338,(undefined (**) [32])local_638);
ppvVar15 = (LPVOID *)local_518;
lVar20 = 5;
do {
```

Picture 5 file extensions

Below is a section of code provided manually to build a path in:

wchar_t (Unicode string).

local_268._0_1_ = 'C'; // The beginning of the string

...

uStack_250._3_1_ = '\\'; // The characters continue.

...

uStack_24c._3_1_ = 'p'; // "beep"

local_248 = 0x7379732e; // ".sys" (using hex values)

In the end, the path is created as path

wchar_t *path = L"C:\\Windows\\System32\\drivers\\beep.sys"

Next, it proceeds to search for logical drives in the system, such as ****C:***, ****D:***, etc.

For each discovered drive, it uses several structures and functions to analyze or process the data on those drives.

```

FUN_14002ab08((LPVOID *)local_5d8);
}
local_5e0 = 0;
while (uVar5 = local_5e0,
      local_5e0 < (ulonglong)((longlong)local_3b8 - (longlong)local_3c0) / 0x18) {
    ppauVar1 = local_3c0 + local_5e0 * 3;
    _local_5d8 = ZEXT816(0);
    local_5e0 = 0;
    DVar7 = GetLogicalDrives();
    local_647 = 0x41;
    while (auVar2 = _local_5d8, bVar4 = local_647, (char)local_647 < '[') {
        if ((DVar7 >> ((int)(char)local_647 - 0x41U & 0x1f) & 1) != 0) {
            _local_638 = (_TOKEN_PRIVILEGES)ZEXT816(0);
            local_628 = 0;
            local_620 = 0;
            FUN_14002ad80((undefined (*) [32])local_638, local_647, 1);
            pauVar9 = FUN_140028040((undefined (*) [32])local_638, (undefined (*) [32])&DAT_14005f2bc, 2);
        };
        local_618 = *(undefined (*) [16])*pauVar9;
    }
}

```

Figure 6 Searching of disks

Function FUN_14002917c

It reads every file and folder in a specified path.

If it finds any file that matches a list of names (filters), it saves it in a list.

If it finds a subfolder, it recursively dives into it.

This entire operation is done with careful manual memory management, which indicates that it is part of the malicious file.

```

pauVar9 = *(undefined (**) [32])*param_1;
}
pauVar16 = param_3;
FUN_1400280b4((undefined (*) [32])local_1a8, param_2, param_3, pauVar9,
              *(ulonglong *)(*param_1 + 0x10), (undefined (*) [32])&DAT_14005f2b8, 2);
lpFindFileData = (LPWIN32_FIND_DATA)&local_168;
lpFileName = local_1a8;
if (0xf < local_190) {
    lpFileName = local_1a8[0];
}
hFindFile = FindFirstFileA((LPCSTR)lpFileName, lpFindFileData);
if (hFindFile != (HANDLE)0xffffffffffffffff) {
    do {
        if ((local_13c[0] != '.') ||
            ((local_13c[1] != '\0' && ((local_13c[1] != '.' || (local_13c[2] != '\0'))))) {
            if (*(ulonglong *)(*param_1 + 0x10) == 0x7fffffffffffffff) {

```

Figure 7 Reading files and directories

Function FUN_140025d40

This function:

1. Opens the Service Control Manager with `OpenSCManagerA(NULL, NULL, 2)` (access for creating and modifying services).
2. Creates a new service named `RAW_IO` using `CreateServiceA(...)`, and for the executable path, it uses `param_1` (a parameter passed from the outside)
RAW_IO is the name of the service (suspicious).
`param_1` is the path of the executable or driver that will be started as a service.
3. Starts the service with `StartServiceA(...)`.
 - o If the start is unsuccessful but `GetLastError() == 0x420`

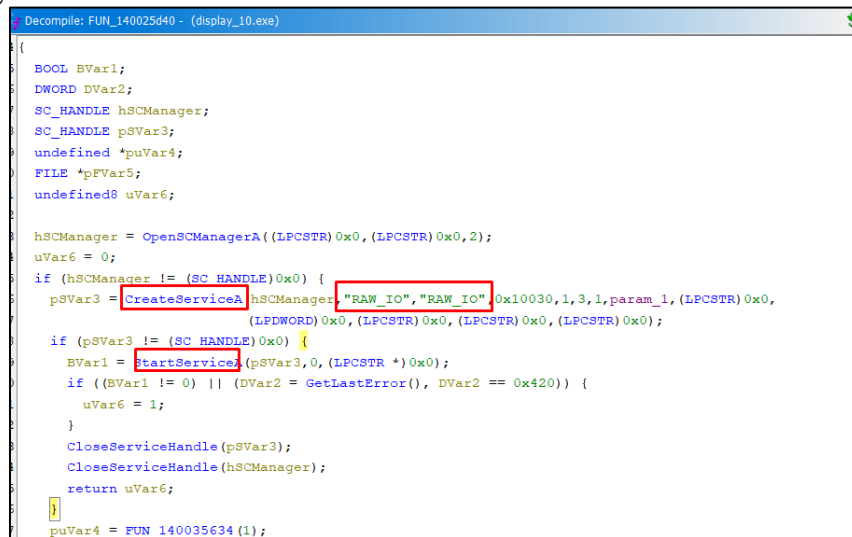
(ERROR_SERVICE_ALREADY_RUNNING), it is still considered a success..

4. If some steps fail, it checks if RAW_IO has been created previously with OpenServiceA(...).

Finally, it returns 1 if successful, 0 if not.

param_1 is the path of the file that will be used as the executable for the service.

Therefore, in this specific case, the **rawio.sys** driver will be the **rawio.sys** driver created in the Temp directory.



```
Decompile: FUN_140025d40 - (display_10.exe)
{
    BOOL BVar1;
    DWORD DVar2;
    SC_HANDLE hSCManager;
    SC_HANDLE pSvc3;
    undefined *puVar4;
    FILE *pFVar5;
    undefined uVar6;

    hSCManager = OpenSCManagerA((LPCSTR)0x0, (LPCSTR)0x0, 2);
    uVar6 = 0;
    if (hSCManager != (SC_HANDLE)0x0) {
        pSvc3 = CreateServiceA(hSCManager, "RAW_IO", "RAW_IO", 0x10030, 1, 3, 1, param_1, (LPCSTR)0x0,
            (LPDWORD)0x0, (LPCSTR)0x0, (LPCSTR)0x0);

        if (pSvc3 != (SC_HANDLE)0x0) {
            BVar1 = StartService(pSvc3, 0, (LPCSTR *)0x0);
            if ((BVar1 != 0) || (DVar2 = GetLastError(), DVar2 == 0x420)) {
                uVar6 = 1;
            }
            CloseServiceHandle(pSvc3);
            CloseServiceHandle(hSCManager);
            return uVar6;
        }
    }
    puVar4 = FUN_140035634(1);
}
```

Figure 8 Creation of the malicious service RAW_IO

The function **FUN_14002b620** is the most critical function of the analysis. It uses a technique for direct disk writing, through a driver called **sectorio.sys**, to modify disk sectors at a low level, manipulate the boot sector, or overwrite **.exe/.dll** files as part of a "wiper" technique, i.e., in data destruction.

- This function uses the **sectorio.sys** driver to perform an **IOCTL** (Input/Output Control) that:
- Allows direct writing to disk sectors (very dangerous).
- Bypasses Windows protections for writing to MBR/Volume Boot Record or directly to protected files.
- Bypasses endpoint protection, avoiding writing through Windows APIs.

```

Decompile: FUN_14002b620 - (display_10.exe)
5  longlong lVar8;
6  undefined8 uVar9;
7  undefined local_res18;
8  undefined3 uStack_133;
9  undefined4 uStack_130;
10 undefined uStack_12c;
11 LPSTR local_128 [2];
12 CHAR local_118 [272];
13
14
15 local_res18 = param_3;
16 hDevice = CreateFileA("\\\\.\\sectorio",0xc0000000,3,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,(HANDLE)
0x0
17
18 );
19 uVar9 = 1;
20 if (hDevice == (HANDLE)0xffffffff) {
21     FUN_14004bcc0((undefined (*) [32])local_118,0,0x104);
22     GetFullPathNameA("sectorio.sys",0x108,local_118,local_128);
23     pvVar6 = CreateFileA(local_118,0x80000000,1,(LPSECURITY_ATTRIBUTES)0x0,3,0x80,(HANDLE)0x0);
24     if (pvVar6 != (HANDLE)0xffffffff) goto LAB_14002b7cd;
25     hDevice = (HANDLE)0x0;
26 }

```

Figure 9 sectorio.sys

During the analysis, a piece of code was also identified in an unnamed function, where a series of file names are encoded, such as **LuCallbackProxy.exe**, **ccScHvSch.exe**, **SmSc.exe** (from 0x6578652e636d53), **AccApp.exe**, **SysmSrv.exe**, **SysmEAFE.exe**, **IR.exe**, **SysclmScvs**. These are all names associated with **Symantec/Norton/Security Suite** executables.

This piece of code is designed to bypass **Symantec** protection by using the names of its legitimate processes, which could prevent the initialization of protective modules (for example, if Symantec uses process name checks for self-protection).

```

Decompile: UndefinedFunction_140001000 - (display_10.exe)
5  uStack_f0 = 0xf;
6  uStack_ec = 0;
7  stack0xfffffffffffff00 = 0x6578652e;
8  auStack_108._0_8_ = 0x7473486376536363;
9  uStack_d8 = 7;
10 uStack_d4 = 0;
11 uStack_d0 = 0xf;
12 uStack_cc = 0;
13 auStack_b8 = ZEXT816(0);
14 uStack_fc = 0;
15 auStack_e8 = ZEXT716(0x6578652e636d53);
16 auStack_c8 = ZEXT816(0);
17 FUN_140027b58((undefined (*) [32])auStack_c8,(undefined (*) [32])"LuCallbackProxy.exe",0x13);
18 uStack_98 = 9;
19 uStack_94 = 0;
20 uStack_90 = 0xf;
21 uStack_8c = 0;
22 stack0xfffffffffffff60 = 0x65;
23 auStack_a8._0_8_ = 0x78652e7070416363;
24 stack0xfffffffffffff80 = 0x6578;
25 auStack_88._0_8_ = 0x652e7672536d7953;
26 stack0xfffffffffffffa0 = 0x6578;
27 auStack_68._0_8_ = 0x652e4146456d7953;

```

Figure 10 Services of symantec ect

The function **FUN_14002b390** is the function used by the malicious file to communicate with its driver **sectorio.sys** and execute RAW disk operations (direct reading/writing to sectors).

- Selects the volume based on driveIndex..
- Verifies that the volume is NTFS (most Windows systems).
- Opens the disk in RAW I/O (bypasses the file system and most AV protections).
- Packs the size/offset (local_e0) and sends it to the driver via the custom IOCTL.
- The driver sectorio.sys executes physical read/write operations and returns the real address of the affected sector.
- Uses a non-standard IOCTL (0x560020) → requests the existence of sectorio.sys.
 - Writes to disk at the sector level → can delete, damage, or hide data without being detected by AVs.
- Verifies NTFS to ensure compatibility with the Windows disk structure.
- Relies on paths "\\X:" – a classic technique used by malware/wipers to bypass high-level APIs

The main functionality of this file is the destruction of Windows operating system files and the user's own files. Therefore, the process works as follows: the legitimate **Windows beep.sys** file is used as input to overwrite files via the **rawio.sys** driver, which is actually **sectorio.sys**. In this case, the role of the driver functions as a Proxy, where the entire overwriting process does not occur from the executable file itself but from the driver itself. The file used for overwriting is the **beep.sys** file, which is used as a parameter to overwrite the files of the operating system itself.

Analysis of the rawio.sys file

The **display_10.exe** file requires administrator privileges to operate, as creating a kernel-type service will require administrative rights. What makes this attack successful is the legitimate certificate it is signed with. The certificate in question is not for the **display_10.exe** file, but for **rawio.sys**, as this is the file that is defined as the binary executable in the service.

On a modern Windows x64 system with Secure Boot / DSE (Driver Signature Enforcement) enabled, **sectorio.sys** without a certificate cannot be loaded as a kernel driver, even though it can be registered as a service. It would start only if the attacker:

- Disables Driver Signature Enforcement (Test Mode, Safe Mode with Disable integrity checks)
- Uses a "bring-your-own-vulnerable-driver" method
- Or it is on a 32-bit system / very old system where signing is not enforced.

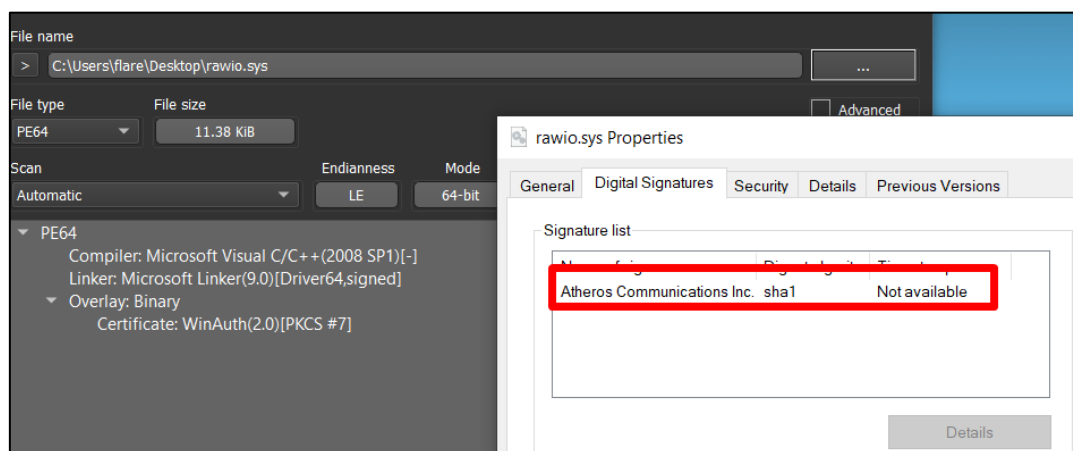


Figure 11 Certificate of rawio.sys

if we query the Windows services with the name **RAW_IO**, the path of **rawio.sys** will also be revealed, which is located in the previously mentioned **Temp** path.

```

C:\> Command Prompt [Link]
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

FLARE-VM Wed 06/25/2025 10:59:44.38
C:\Users\flare>sc qc RAW_IO
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: RAW_IO
        TYPE               : 1  KERNEL_DRIVER
        START_NAME           : 3  DEMAND_START
        ERROR_CONTROL        : 1  NORMAL
        BINARY_PATH_NAME     : \??\C:\Users\flare\AppData\Local\Temp\rawio.sys
        LOAD_ORDER_GROUP     :
        TAG                  : 0
        DISPLAY_NAME         : RAW_IO
        DEPENDENCIES         :
        SERVICE_START_NAME   :

FLARE-VM Wed 06/25/2025 10:59:48.05
C:\Users\flare>

```

Figure 12 The RAW_IO.sys service

During the identification and threat search, a GitHub repository was found, which was used to create this driver: <https://github.com/jschicht/SectorIo/>. From the code written in C, it was understood that this Windows x64 (kernel-mode) driver exposes an object called **\Device\sectorio** → **\DosDevices\sectorio** and allows user-mode processes to perform RAW read/write operations on the sectors of any disk or partition existing on the system.

```

WCHAR g_szDeviceName[] = L"\\Device\\sectorio";
WCHAR g_szDosDeviceName[] = L"\\DosDevices\\sectorio";

```

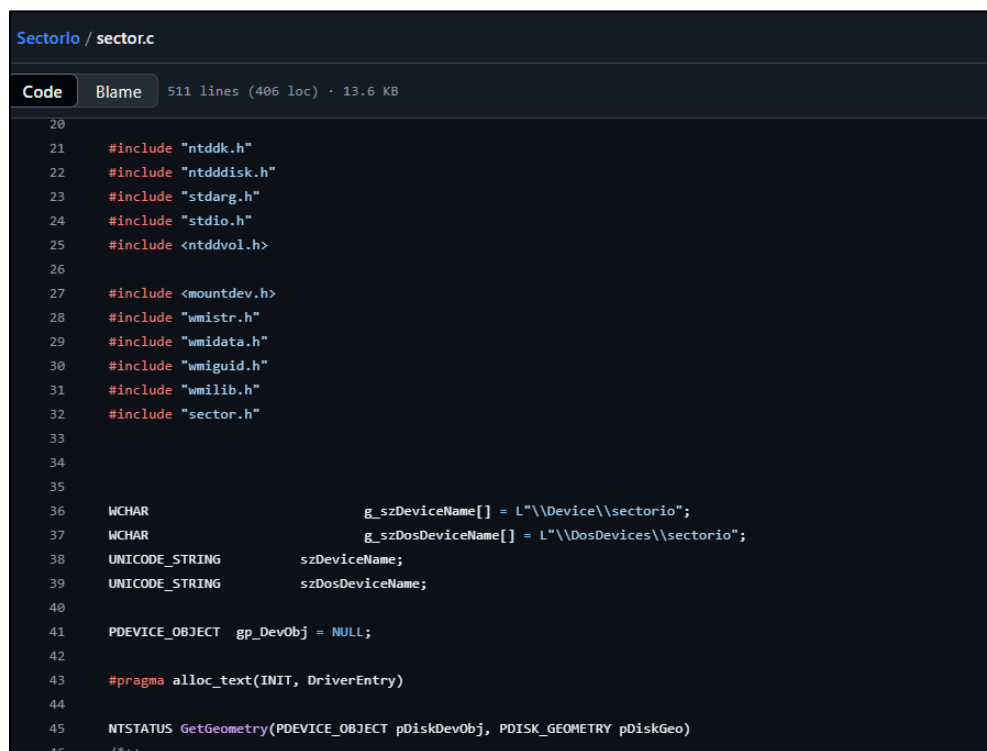
```

IoCreateDevice(..., &szDeviceName, FILE_DEVICE_UNKNOWN, ..., &gp_DevObj);

```

IoCreateSymbolicLink(&szDosDeviceName, &szDeviceName);

sectorio.sys exposes a device \\sectorio that accepts an IOCTL (0x560020) to read/write physical sectors on any NTFS disk/partition. The driver enumerates all disk.sys objects and stores the sector size for each. Any user-mode process that opens the device can call IOCTL_SECTOR_WRITE and perform direct disk write (RAW) without any privilege verification, bypassing file integrity mechanisms and antivirus monitoring. This makes the driver an ideal tool for wipers, rootkits, or ransomware that want to manipulate the disk beneath the file system level.



```
Sectorio / sector.c
Code Blame 511 lines (406 loc) · 13.6 KB
20
21 #include "ntddk.h"
22 #include "ntdddisk.h"
23 #include "stdarg.h"
24 #include "stdio.h"
25 #include <ntddvol.h>
26
27 #include <mountdev.h>
28 #include "wmistr.h"
29 #include "wmidata.h"
30 #include "wmiguid.h"
31 #include "wmilib.h"
32 #include "sector.h"
33
34
35
36 WCHAR g_szDeviceName[] = L"\\Device\\sectorio";
37 WCHAR g_szDosDeviceName[] = L"\\DosDevices\\sectorio";
38 UNICODE_STRING szDeviceName;
39 UNICODE_STRING szDosDeviceName;
40
41 PDEVICE_OBJECT gp_DevObj = NULL;
42
43 #pragma alloc_text(INIT, DriverEntry)
44
45 NTSTATUS GetGeometry(PDEVICE_OBJECT pDiskDevObj, PDISK_GEOMETRY pDiskGeo)
46 /*++
```

Picture 13 source code of sectorio.c



Figure 14 Post display_10.exe execution

Indicators of Compromise

81EB22828306F3197B35FEF2035CEF2C548F587F8511902852964850023389D7	Display_10.exe
06B7A3CD3266449294EEEFB957965EA9F1354804696955E1EB1A7F9B515F5880	Rawio.sys

Recommendations

The National Cybersecurity Authority recommends:

- Open Command Prompt as administrator and type:
sc qc RAW_IO
- Then, search for rawio.sys and delete it from the directory where it is located.
- Restore the system from a backup or reinstall Windows if your system has been affected by this virus.
- Immediately block the Indicators of Compromise (IOCs) mentioned above, using your protective security devices.
- Conduct behavioral-level traffic analysis on endpoint devices by applying EDR (Endpoint Detection and Response) and XDR (Extended Detection and Response) solutions. This enables malicious file analysis not only at the signature level but also based on behavioral patterns.
- Train non-technical staff on phishing attacks and best practices to avoid infection.