

📁	README.assets		
📄	COOLit.sys		
📄	LenovoPCManagerS...		
📄	README.md		
📄	poc.c		

📖 README

CVE-Publication

Vulnerability Title:Lenovo Thermal Master Incorrect Access Control

Affected versions:1.3.5

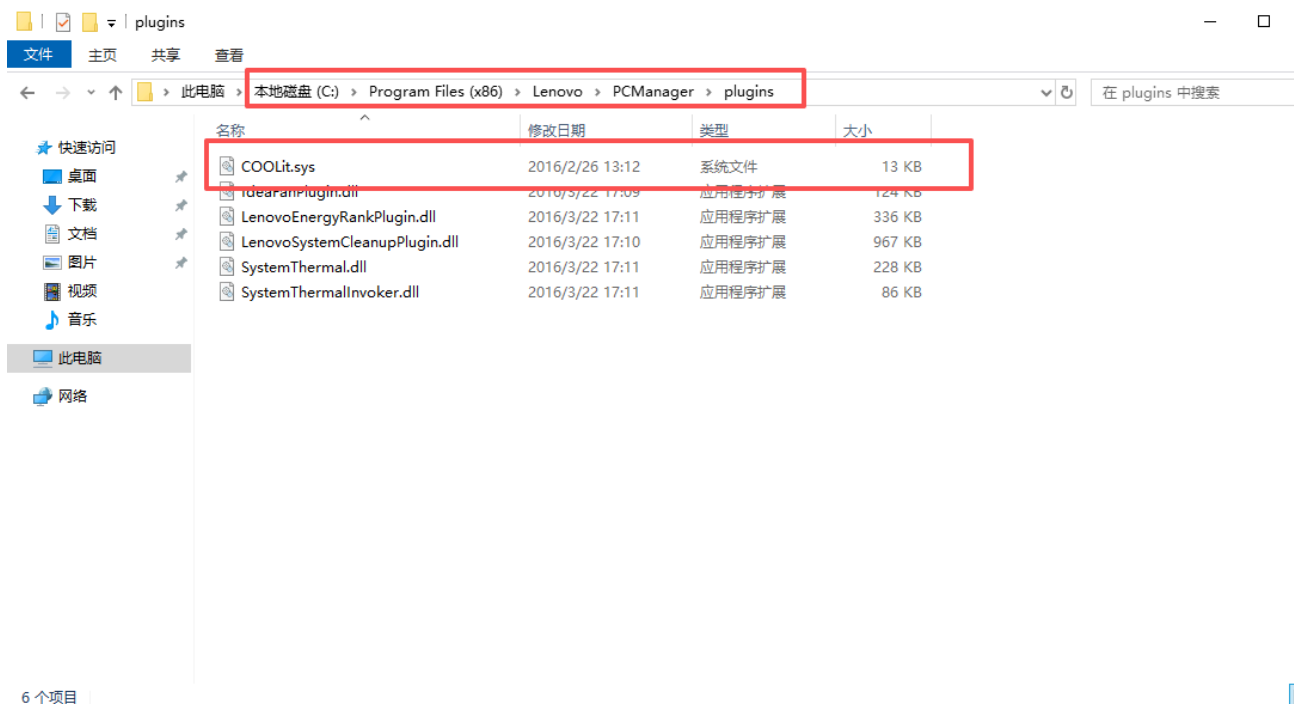
Discovery time:December 21, 2025

Discoverer:ZhouRui

Analysis Report:

Lenovo Thermal Master is a user-friendly computer temperature monitoring software with a clean interface. The official version of Lenovo Thermal Master can detect the temperature of your computer's CPU, motherboard, and hard drive. It is an official computer hardware temperature monitoring manager released by Lenovo, which has functions such as optimizing power consumption and cleaning up system junk files, easily helping you cool down your hardware.

Running this application loads the COOLit.sys driver. An ordinary user can abuse the driver's IoControlCode `0x9C402040` and `0x9C402084` to read arbitrary MSR register values.



The driver exposes a symbolic link(`\\DosDevices\\WinRing0_1_2_0`), which allows application-level programs to call the driver to send IOCTL operations.

```

1 void __fastcall sub_11328(__int64 a1)
2 {
3     struct _DEVICE_OBJECT *v1; // rbx
4     struct _UNICODE_STRING DestinationString; // [rsp+20h] [rbp-18h] BYREF
5
6     v1 = (struct _DEVICE_OBJECT *)0;
7     RtlInitUnicodeString(&DestinationString, L"\\DosDevices\\WinRing0_1_2_0");
8     IoDeleteSymbolicLink(&DestinationString);
9     if ( v1 )
10        IoDeleteDevice(v1);
11 }

```

First, an IOCTL operation with IoControlCode `0x9C402040` is sent, and the passed SystemBuffer is `0x0000000000000000`. This ensures that the if statement can be entered, which allows `byte_13114` to be assigned the value 1, which is very important.

```
44     if ( !byte_13114 )
45         goto LABEL_24;
46     *(_DWORD *)Irp->AssociatedIrp.MasterIrp = dword_13110;
47 LABEL_29:
48     *p_Information = 4i64;
49     goto LABEL_37;
50     case 0x9C402084:
51         if ( CurrentStackLocation->Parameters.Create.Options >= 8 )
52             {
53                 SystemBuffer = *(_QWORD *)Irp->AssociatedIrp.MasterIrp;
54                 if ( ((unsigned __int8)SystemBuffer ^ (unsigned __int8)BYTE2(SystemBuffer) ^ BYTE4(SystemBuffer) ^ BYTE6(SystemBuffer))) == (BYTE1(S
55                     {
56                         byte_13114 = 1;
57                         goto LABEL_37;
58                     }
59             )
60 LABEL_24:
61     v5 = 0xC0000142;
62     break;
63     case 0x9C402084:
64         if ( byte_13114 )
65             {
66                 v5 = sub_1136C(
67                     (unsigned int *)Irp->AssociatedIrp.MasterIrp,
68                     CurrentStackLocation->Parameters.Create.Options,
69                     (unsigned __int64 *)Irp->AssociatedIrp.MasterIrp,
70                     CurrentStackLocation->Parameters.Read.Length,
71                     p_Information);
72                 break;
73             }
```

In this way, when an IOCTL operation with IoControlCode `0x9C402084` is sent, the `sub_1136C` function can be executed by checking.

```
57         goto LABEL_37;
58     }
59 }
60 LABEL_24:
61     v5 = 0xC0000142;
62     break;
63     case 0x9C402084:
64         if ( byte_13114 )
65             {
66                 v5 = sub_1136C(
67                     (unsigned int *)Irp->AssociatedIrp.MasterIrp,
68                     CurrentStackLocation->Parameters.Create.Options,
69                     (unsigned __int64 *)Irp->AssociatedIrp.MasterIrp,
70                     CurrentStackLocation->Parameters.Read.Length,
71                     p_Information);
72                 break;
73             }
74     goto LABEL_24;
75     case 0x9C406144:
76         if ( byte_13114 )
77             {
78                 Length = CurrentStackLocation->Parameters.Read.Length;
79                 if ( CurrentStackLocation->Parameters.Create.Options != 8 )
80                     {
81                         v5 = 0xC0000000;
82                         break;
83                     }
84                 MasterIrp = (ULONG *)Irp->AssociatedIrp.MasterIrp;
85                 BusDataByOffset = HalGetBusDataByOffset(
86                     PCIConfiguration
```

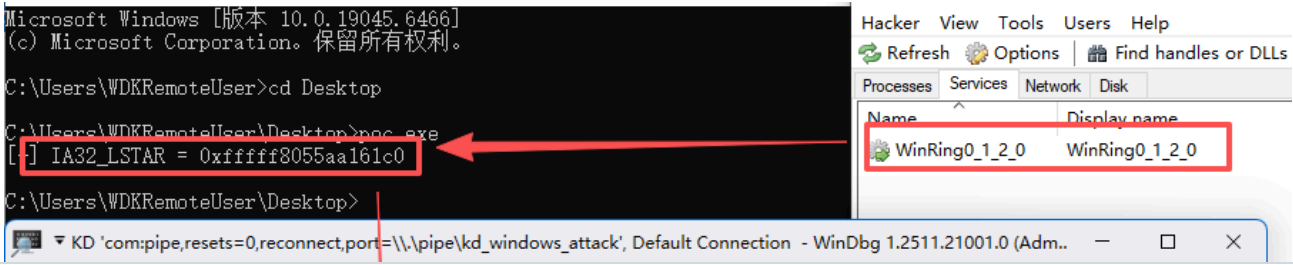
The core of the vulnerability lies in the fact that the `readmsr` function reads the register value represented by the passed `SystemBuffer`.

```

1  __int64 __fastcall sub_1136C(
2      unsigned int *SystemBuffer,
3      __int64 InputBufferLength,
4      unsigned __int64 *SystemBuffer_1,
5      __int64 a4,
6      _DWORD *p_Information)
7  {
8      unsigned __int64 v5; // rax
9
10     v5 = __readmsr(*SystemBuffer);
11     *SystemBuffer_1 = ((unsigned __int64)HIDWORD(v5) << 32) | (unsigned int)v5;
12     *p_Information = 8;
13     return 0i64;
14 }

```

After the driver is loaded, running the poc (without administrator privileges) can read the value of the IA32_LSTAR register, which contains the address of the syscall. This can be verified by using WinDbg, and of course, it can read any MSR register.




Releases

No releases published

Packages

No packages published

Contributors 1

 **CDipper** CDipper

Languages

- C 100.0%