Talos Vulnerability Report

TALOS-2020-1114

NZXT CAM WinRing0x64 Driver IRP 0x9c406104 information disclosure vulnerability

DECEMBER 16, 2020

CVE NUMBER

CVE-2020-13517

Summary

An information disclosure vulnerability exists in the WinRing0x64 Driver IRP 0x9c406104 functionality of NZXT CAM 4.8.0. A specially crafted I/O request packet (IRP) can cause the disclosure of sensitive information. An attacker can send a malicious IRP to trigger this vulnerability.

Tested Versions

NZXT CAM 4.8.0

Product URLs

https://www.nzxt.com/camapp

CVSSv3 Score

6.5 - CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N

CWE

CWE-269 - Improper Privilege Management

Details

NZXT CAM is software designed as an all-in-one solution for computer hardware monitoring and performance. The software monitors fan speeds, CPU temperatures, network and RAM usage, as well as CPU/GPU frequencies for overclocking. It also has features for in-game overlays to track PC performance. The software also has an inventory for all devices that are installed on the PC at any given time.

The WinRing0x64 driver exists so that the NZXT CAM software can have access to the Windows Kernel as well as elevated privileges required to talk to PCI devices as well as making CPU/GPU configuration changes. This driver creates `\Device\WinRing0_1_2_0` that is accessible to any user on the system and this driver is used for all elevated tasks.

Using the IRP 0x9c406104 gives a low privilege user direct access to the `MmMapIoSpace` function that can read memory between 0xC_0000 and 0xF_FFFF. The memory pointed to by `MmMapIoSpace` is physical memory of the device, this location in physical memory often stores BIOS information or the motherboards firmware. This access could be used for leak sensitive information.

```
00011520  uint64_t rbx = zx.q(arg4)
0001151d  int64_t rax_6
0001151d  if (arg2 != 0x10)
000115c1  label_115c1:
000115c1      rax_6 = 0xc000000d
00011532  else
00011532      uint64_t rax_2 = zx.q(*(arg1 + 0xc) * *(arg1 + 8))
00011538      if (rbx:0.d u< rax_2:0.d)
00011538          goto label_115c1
0001153e      int64_t rcx = *arg1
00011548      if (rcx s< 0xc0000)
00011548          goto label_115c1
0001154a      uint64_t rbp_1 = zx.q(rax_2:0.d)
00011557      if (rcx + rax_2 - 1 s> 0xfffff)
00011557          goto label_115c1
0001155f      int64_t rax_4 = MmMapIoSpace(rcx, rbp_1, 0)
00011565      uint64_t rcx_1 = zx.q(*(arg1 + 8))
00011568      int64_t r12
00011568      r12:0.b = 0
0001156b      uint64_t rcx_2 = zx.q(rcx_1:0.d - 1)
0001156b      if (rcx_1:0.d == 1)
0001159a          uint64_t rcx_6 = zx.q(*(arg1 + 0xc))
0001159d          int32_t* rdi_3 = arg3
000115a0          int32_t* rsi_3 = rax_4
000115a3          for (; rcx_6 != 0; rcx_6 = rcx_6 - 1)
000115a3              *rdi_3 = *rsi_3
000115a3              rdi_3 = rdi_3 + 1
000115a3              rsi_3 = rsi_3 + 1
00011570      else
00011570          uint64_t rcx_3 = zx.q(rcx_2:0.d - 1)
00011570          if (rcx_2:0.d == 1)
0001158c              uint64_t rcx_5 = zx.q(*(arg1 + 0xc))
0001158f              int32_t* rdi_2 = arg3
00011592              int32_t* rsi_2 = rax_4
00011595              for (; rcx_5 != 0; rcx_5 = rcx_5 - 1)
00011595                  *rdi_2 = *rsi_2
00011595                  rdi_2 = rdi_2 + 2
00011595                  rsi_2 = rsi_2 + 2
00011575          else if (rcx_3:0.d != 2)
0001157a              r12:0.b = 1
0001157f          else
0001157f              uint64_t rcx_4 = zx.q(*(arg1 + 0xc))
00011582              int32_t* rdi_1 = arg3
00011585              int64_t rsi_1 = rax_4
00011588              for (; rcx_4 != 0; rcx_4 = rcx_4 - 1)
00011588                  *rdi_1 = *rsi_1
00011588                  rdi_1 = rdi_1 + 4
00011588                  rsi_1 = rsi_1 + 4
000115ab      MmUnmapIoSpace(rax_4, rbp_1)
000115b4      if (r12:0.b != 0)
000115b4          goto label_115c1
000115bb      *arg5 = rbx:0.d
000115bd      rax_6 = 0
000115e0  return rax_6
```

Exploit Proof of Concept

This proof of concept reads 0x1_0000 bytes of data starting from 0xF_0000 of physical memory. This is the beginning of the motherboard bios.

```
[+] Getting Device Driver Handle
                [+] Device Name: \\.\WinRing0_1_2_0
                [+] Device Handle: 0x94
        [+] Setting Up Vulnerability Stage
                [+] Allocating Memory For Buffer
                        [+] Memory Allocated: 0x0000026669513F50
                        [+] Allocation Size: 0x10
                [+] Preparing Buffer Memory Layout
000F0000 00000000 00000004 00004000
0000CA9E 00000000 00000000 00000000 24454649 000253B9 00008307 010BF000 0C00F000 DA438018 00000000 000F05A0 6F400000 0198000E C7800000
0120000F 00000000 00000000 E4000000 00000000 E0000000 00004004 00000000 00000000 000FF800 00000000 FFFFFFFF...[output truncated]
```

Timeline

2020-07-17 - Vendor Disclosure

2020-08-10 - Vendor acknowledged; Talos issued copy of reports

2020-11-30 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT NEXT REPORT

TALOS-2020-1113 TALOS-2020-1115