ZEROPERIL

VULNERABILITY REPORT

[Vendor: AMD]

[Product: AMD Chipset Drivers]

[Author: Kyriakos Economou]

[08/04/2021]

OVERVIEW

ZeroPeril Ltd has discovered two issues inside the *amdpsp.sys* (v4.13.0.0) kernel driver module that ships with the *AMD Chipset Drivers* package for multiple AMD chipsets [1]:

- B350
- A320
- X370
- X399
- B450
- X470
- X570
- B550
- A520
- TRX40
- WRX80

The first issue is an **information disclosure** type security vulnerability, and the second is a **memory leak** type bug due to insufficient releasing of all associated allocated resources upon request.

We have verified both in the latest *Revision Number* (2.13.27.501) of the package that was released the 4th of February 2021.

TECHNICAL DETAILS

The amdpsp.sys driver exposes one or more named device objects of the form *\Device\amdpsp* in userland to allow user-mode processes to send I/O control (IOCTL) requests. The DACL applied to the device object allows also low privileged users to open a handle and send requests to the driver.

We have identified two IOCTLs of interest that are related to these findings:

- *0x9c422008*
- *0x9c42200c*

For simplicity, we will be referring to the first one as *ALLOC_CONTIGUOUS_MEMORY* and to the second as *FREE_CONTIGUOUS_MEMORY* control codes (CTLs).

With regards to the first issue, when the driver receives an ALLOC_CONTIGUOUS_MEMORY request, it will make a call to *MmAllocateContiguousMemorySpecifyCache* function [2] to allocate a range of contiguous physical memory and map it to nonpaged kernel pool address space. Next, the driver will attempt to initialize the allocated range with zeros by using the exact allocation size as requested.

```
.text:0000000140002B82 loc_140002B82:                          ; DATA XREF: .rdata:0000000140009624↓o
.text:0000000140002B82                                         ; .rdata:000000014000963C↓o ...
.text:0000000140002B82                 mov     [rsp+38h+arg_0], rbp
.text:0000000140002B87                 mov     ebp, edx
.text:0000000140002B89                 mov     r8, 0FFFFFFFFFh
.text:0000000140002B93                 mov     r9d, 4000000h
.text:0000000140002B99                 mov     edx, ebx
.text:0000000140002B9B                 mov     ecx, ebp
.text:0000000140002B9D                 mov     [rsp+38h+arg_8], rdi
.text:0000000140002BA2                 mov     [rsp+38h+var_18], 1
.text:0000000140002BAA                 call    cs:MmAllocateContiguousMemorySpecifyCache
.text:0000000140002BB0                 mov     rdi, rax
.text:0000000140002BB3                 test    rax, rax
.text:0000000140002BB6                 jnz     short loc_140002BDA
.text:0000000140002BB8                 mov     rdi, [rsp+38h+arg_8]
.text:0000000140002BBD                 mov     rbp, [rsp+38h+arg_0]
.text:0000000140002BC2                 mov     ebx, 0C000000Dh
.text:0000000140002BC7                 mov     eax, ebx
.text:0000000140002BC9                 mov     rbx, [rsp+38h+arg_10]
.text:0000000140002BCE                 mov     rsi, [rsp+38h+arg_18]
.text:0000000140002BD3                 add     rsp, 30h
.text:0000000140002BD7                 pop     r14
.text:0000000140002BD9                 retn
.text:0000000140002BDA ; ---------------------------------------------------------------------------
.text:0000000140002BDA
.text:0000000140002BDA loc_140002BDA:                          ; CODE XREF: sub_140002B50+66↑j
.text:0000000140002BDA                                         ; DATA XREF: .pdata:000000014001C138↓o
.text:0000000140002BDA                 mov     r8, rbp         ; Size
.text:0000000140002BDD                 xor     edx, edx        ; Val
.text:0000000140002BDF                 mov     rcx, rax        ; Dst
.text:0000000140002BE2                 call    memset
```

Figure 1. Memory allocation and initialisation

The first part of the issues starts with memory contents initialisation. The size of memory allocated via *MmAllocateContiguousMemorySpecifyCache* function is rounded to the size of memory page as defined by the system, which is usually 4KBs (4096 bytes) of memory.

In other words, if the request sent to the driver asks for 1-byte allocation, the function will still allocate an entire memory page, but will only initialise the first byte, since it doesn't take in consideration the memory allocation granularity. What this means is that the rest of data on that page will remain intact.

Following that step, the driver will call *MmMapLockedPagesSpecifyCache* [3] using *UserMode* access mode which will map that kernel nonpaged pool page in userland. This allows the process to parse the non-initialised contents and retrieve information that otherwise would only be accessible by high-privileged processes and/or code running at kernel level.

```
.text:0000000140003099          mov     rcx, [rsp+68h+VirtualAddress] ; VirtualAddress
.text:000000014000309E          mov     esi, eax
.text:00000001400030A0          test    rcx, rcx
.text:00000001400030A3          jz      loc_140003198
.text:00000001400030A9          xor     r9d, r9d        ; ChargeQuota
.text:00000001400030AC          xor     r8d, r8d        ; SecondaryBuffer
.text:00000001400030AF          mov     edx, r14d       ; Length
.text:00000001400030B2          mov     [rsp+68h+Irp], r13 ; Irp
.text:00000001400030B7          call    cs:IoAllocateMdl
.text:00000001400030BD          mov     rbp, rax
.text:00000001400030C0          test    rax, rax
.text:00000001400030C3          jz      loc_14000318A
.text:00000001400030C9          mov     rcx, rax        ; MemoryDescriptorList
.text:00000001400030CC          call    cs:MmBuildMdlForNonPagedPool
.text:00000001400030D2          mov     r8d, 1          ; CacheType
.text:00000001400030D8          xor     r9d, r9d        ; BaseAddress
.text:00000001400030DB          mov     rcx, rbp        ; MemoryDescriptorList
.text:00000001400030DE          movzx   edx, r8b        ; AccessMode
.text:00000001400030E2          mov     [rsp+68h+Priority], 40000000h ; Priority
.text:00000001400030EA          mov     dword ptr [rsp+68h+Irp], r13d ; BugCheckOnFailure
.text:00000001400030EF          call    cs:MmMapLockedPagesSpecifyCache
```

Figure 2. Usermode mapping of nonpaged kernel pool

The amdpsp.sys driver has an array containing a maximum of 100 mappings at the same time.

Each userland mapping base address will be stored inside that array so that can be retrieved for a subsequent operation.

We can now proceed to the second issue that involves a *FREE_CONTIGUOUS_MEMORY* request to the driver.

Upon receiving this request, the driver will extract the userland base address for the mapping that is supplied via the aforementioned CTL code, and will search through the list of stored mappings.

Once the correct entry has been found, it will call *MmFreeContiguousMemory* [4] to release the kernel nonpaged pool allocation that is associated with that mapping. According to the documentation, this should release the range of physically contiguous memory that was previously allocated.

```
.text:FFFFF80440CE2E20          sub     rsp, 28h
.text:FFFFF80440CE2E24          test    rcx, rcx
.text:FFFFF80440CE2E27          jz      short loc_FFFFF80440CE2E3A
.text:FFFFF80440CE2E29          test    edx, edx
.text:FFFFF80440CE2E2B          jz      short loc_FFFFF80440CE2E3A
.text:FFFFF80440CE2E2D          call    cs:MmFreeContiguousMemory
.text:FFFFF80440CE2E33          xor     eax, eax
.text:FFFFF80440CE2E35          add     rsp, 28h
.text:FFFFF80440CE2E39          retn
```

Figure 3. Releasing allocated physical pages

However, even though the nonpaged pool allocation is freed, the driver never calls *MmUnmapLockedPages [5]* which results into keeping the mapping of those physical pages in user-mode and subsequently keep them private to the associated process until it’s terminated. This means that these physical pages become unusable by the system for the lifespan of the process.

Furthermore, additional memory pool chunks allocated for other kernel objects during the memory mapping stage such as *MDL* objects [6] also are not freed until the calling process has been terminated, which increases the memory leak issue by rendering additional memory unusable for other system operations.

SECURITY IMPACT

During our tests we managed to leak several gigabytes of uninitialized physical pages by allocating and freeing blocks of 100 allocations continuously until the system was not able to return a contiguous physical page buffer.

The contents of those physical pages varied from kernel objects and arbitrary pool addresses that can be used to circumvent exploitation mitigations such as *KASLR* [7], and even registry key mappings of *\Registry\Machine\SAM* containing *NTLM* hashes [8] of user authentication credentials that can be used in subsequent attack stages.

For example, these can be used to steal credentials of a user with administrative privilege and/or be used in *pass-the-hash* [9] style attacks to gain further access inside a network.

MITIGATION ADVICE

1. Use appropriate DACLs on device objects to block non-privileged users from sending IOCTL requests to a kernel driver whenever possible
2. Avoid userland mappings of kernel pool memory.
3. If mapping kernel pool memory in userland is necessary due to the current design, then make sure that the memory has been initialised appropriately.
4. Always make sure that allocated resources are freed back to the system when no longer in use.

REFERENCES

1. https://www.amd.com/en/support/chipsets/amd-socket-tr4/x399
2. https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-mmallocatecontiguousmemoryspecifycache
3. https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-mmmaplockedpagesspecifycache
4. https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-mmfreecontiguousmemory
5. https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-mmunmaplockedpages
6. https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/using-mdls
7. https://docs.microsoft.com/en-us/windows/security/threat-protection/overview-of-threat-mitigations-in-windows-10#table-2
8. https://docs.microsoft.com/en-us/windows/win32/secauthn/microsoft-ntlm
9. https://www.sans.org/reading-room/whitepapers/testing/pass-the-hash-attacks-tools-mitigation-33283