Talos Vulnerability Report

TALOS-2022-1527

ESTsoft Alyac OLE header parsing integer overflow

AUGUST 3, 2022

CVE NUMBER

CVE-2022-32543

SUMMARY

An integer overflow vulnerability exists in the way ESTsoft Alyac 2.5.8.544 parses OLE files. A specially-crafted OLE file can lead to a heap buffer overflow which can result in arbitrary code execution. An attacker can provide a malicious file to trigger this vulnerability.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

ESTsoft Alyac 2.5.8.544

PRODUCT URLS

Alyac - https://www.estsecurity.com/public/product/alyac

CVSSV3 SCORE

7.3 - CVSS:3.0/AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H

CWE

CWE-680 - Integer Overflow to Buffer Overflow

DETAILS

Alyac is an antivirus program for Microsoft Windows, developed by ESTsecurity, which is part of ESTsoft.

When Alyac is scanning an OLE formatted file with signature `D0 CF 11 E0 A1 B1 1A E1`, it executes function `sub_18007F7B0` to check the file signature and bytes that follow.

As a result, either `0x200` or `0x1000` is stored at offset `+0x60` of the OLE parser object [1,2]. This value is used later when calculating the size of heap allocation.

```
; ... file checks ...
.text:000000018007F879                movzx   ecx, word ptr [r8+1Ah]          ; R8
points to the base of the file
.text:000000018007F87E                lea     eax, [rcx-3]
.text:000000018007F881                cmp     ax, 1
.text:000000018007F885                ja      short loc_18007F8E2
.text:000000018007F887                mov     eax, 0FFFEh
.text:000000018007F88C                cmp     [r8+1Ch], ax
.text:000000018007F891                jnz     short loc_18007F8E2
.text:000000018007F893                cmp     cx, 3
.text:000000018007F897                jnz     short loc_18007F8AA
.text:000000018007F899                cmp     word ptr [r8+1Eh], 9
.text:000000018007F89F                jnz     short loc_18007F8AA
.text:000000018007F8A1                mov     dword ptr [rdi+60h], 200h       ;
[1] ole parser object + 0x60
.text:000000018007F8A8                jmp     short loc_18007F8BF
.text:000000018007F8AA ; ---------------------------------------------------------
---------------
.text:000000018007F8AA
.text:000000018007F8AA loc_18007F8AA:                         ; CODE XREF:
sub_18007F7B0+E7↑j
.text:000000018007F8AA                                        ; sub_18007F7B0+EF↑j
.text:000000018007F8AA                cmp     cx, 4
.text:000000018007F8AE                jnz     short loc_18007F8E2
.text:000000018007F8B0                cmp     word ptr [r8+1Eh], 0Ch
.text:000000018007F8B6                jnz     short loc_18007F8E2
.text:000000018007F8B8                mov     dword ptr [rdi+60h], 1000h      ;
[2] ole parser object + 0x60
```

If the file is constructed as expected, it calls `sub_180080150` which then quickly calls `sub_18007F9E0`.

```
.text:000000018007F9E0 sub_18007F9E0   proc near               ; CODE XREF:
sub_180080150+9↓p
.text:000000018007F9E0                                         ; DATA XREF:
.rdata:0000000180404750↓o ...
.text:000000018007F9E0
.text:000000018007F9E0 Src             = byte ptr -1028h
.text:000000018007F9E0 var_28          = qword ptr -28h
.text:000000018007F9E0 arg_8           = qword ptr  10h
.text:000000018007F9E0 arg_10          = qword ptr  18h
.text:000000018007F9E0 arg_18          = qword ptr  20h
.text:000000018007F9E0
.text:000000018007F9E0 ; __unwind { // __GSHandlerCheck
.text:000000018007F9E0                 push    rbx
.text:000000018007F9E2                 push    rbp
.text:000000018007F9E3                 push    r15
.text:000000018007F9E5                 mov     eax, 1030h
.text:000000018007F9EA                 call    _alloca_probe
.text:000000018007F9EF                 sub     rsp, rax
.text:000000018007F9F2                 mov     rax, cs:__security_cookie
.text:000000018007F9F9                 xor     rax, rsp
.text:000000018007F9FC                 mov     [rsp+1048h+var_28], rax
.text:000000018007FA04                 mov     rax, [rcx+20h]
.text:000000018007FA08                 mov     rbx, rcx
.text:000000018007FA0B                 xor     ebp, ebp
.text:000000018007FA0D                 mov     ecx, [rax+2Ch]  ; [3] value at file
offset +0x2C
.text:000000018007FA10                 imul    ecx, [rbx+60h]  ; [4] Size which was
set in sub_18007F7B0 e.g 0x200
.text:000000018007FA14                 cmp     rcx, [rbx+18h]  ; [5] size of the
file
.text:000000018007FA18                 jbe     short loc_18007FA22
.text:000000018007FA1A                 lea     eax, [rbp+6]
.text:000000018007FA1D                 jmp     loc_18007FB30
.text:000000018007FA22 ; ---------------------------------------------------------
----------------
.text:000000018007FA22
.text:000000018007FA22 loc_18007FA22:                          ; CODE XREF:
PARSE__18007F9E0+38↑j
.text:000000018007FA22                 call    j_??2@YAPEAX_K@Z ; operator
new(unsigned __int64)    ; [6]
.text:000000018007FA27                 mov     [rbx+40h], rax
.text:000000018007FA2B                 test    rax, rax
.text:000000018007FA2E                 jnz     short loc_18007FA38
.text:000000018007FA30                 lea     ebp, [rax+3]
.text:000000018007FA33                 jmp     loc_18007FB2E
```

`sub_18007F9E0` checks whether value [3] multiplied by [4] is smaller than the size of the file [5]. Here, the multiplication result is truncated to a 32-bit value (being stored to `ECX`) causing integer overflow. This overflowed size value, which is smaller than what is actually needed, is used to allocate heap memory at [6]. So having a large value at [3] will cause integer overflow and the check with the file size will pass.

Then it will do a repeated memcpy of 0x200 bytes (in the case of the crashing testcase) from file to allocated heap memory up to the number of times specified by the value at file offset `+0x2C` [8]. Since the size of the allocate heap memory is not large enough to store `0x200 * [file + 0x2C]` bytes, heap overflow will occur.

```
.text:000000018007FAD4                imul    edi, esi
.text:000000018007FAD7                lea     rdx, [rsp+1048h+Src] ; Src
.text:000000018007FADC                mov     r8d, r14d       ; Size
.text:000000018007FADF                add     rdi, [rbx+40h]
.text:000000018007FAE3                mov     rcx, rdi        ; void *
.text:000000018007FAE6                call    memcpy          ; [7] CRASH!!!!
.text:000000018007FAEB                test    rdi, rdi
.text:000000018007FAEE                jz      short loc_18007FB04
.text:000000018007FAF0                mov     rax, [rbx+20h]
.text:000000018007FAF4                inc     esi
.text:000000018007FAF6                cmp     esi, [rax+2Ch]  ; [8]
```

Following is the stack trace when heap overflow occurs.

```
0:000> k
# Child-SP          RetAddr               Call Site
00 000000a7`6c57e298 00007ffa`adc2faeb     VCRUNTIME140!memcpy+0x1e3
01 000000a7`6c57e2a0 00007ffa`adc3015e     coen!Coen_Clean+0x6bb5b
02 000000a7`6c57f2f0 00007ffa`adc2f8d6     coen!Coen_Clean+0x6c1ce
03 000000a7`6c57f320 00007ffa`adc257fa     coen!Coen_Clean+0x6b946
04 000000a7`6c57f360 00007ffa`adbc91c2     coen!Coen_Clean+0x6186a
05 000000a7`6c57f6a0 00007ffa`adbb5795     coen!Coen_Clean+0x5232
06 000000a7`6c57f820 00007ffa`adbc3974     coen+0x5795
07 000000a7`6c57f960 00007ff7`ecb8116b     coen!Coen_ScanPath+0xb4
...
```

TIMELINE

2022-06-22 - Vendor Disclosure

2022-08-03 - Public Release

2022-08-03 - Vendor Patch Release

CREDIT

Discovered by Jaewon Min of Cisco Talos.