alwil avast! Anti-virus Engine Remote/Local Heap Overflow

04-July-2006

Summary

alwil software produces an anti-virus engine. The engine is capable of scanning diverse archive formats, is purported to detect 100% of in-the-wild viruses, and is ICSA certified. This engine is provided to OEM partners in order to enable their products to scan for viruses.

While processing LHA archives, the software has insufficient checks on the data taken as input from the file. Specifically, the flaw occurs when combining the filename and directory-name extended-header fields of LHA files. This flaw allows a specially crafted LHA file to cause a heap-overflow in the affected software.

Impact

This vulnerability is present by default in alwil's avast! anti-virus engine. Successful exploitation of these vulnerabilities results in local and remote code execution with the full privileges of the process. By default, the privileges are equivalent to System. Exploits that leverage this vulnerability must be lowercase-conversion resistant and not contain NULL bytes; two tractable constraints. Thus, exploitation can be made to work reliably.

Affected software

All products that contain versions of alwil's anti-virus engine less than Version 4.7.869 for desktops or less than version 4.7.660 for servers including:
alwil software: avast! Anti-Virus
alwil software: avast! Server Edition
TN North Software: Interner Anywhere eMailServer
IceWarp Software - Merak Email Server
SmartMax Software, Inc. - MailMax Server
Paul Smith Computer Services - VPOP3 Email Server
NetWin Software - SurgeMail Email Server
Bains Digital - Defender MX

Credit

This vulnerability was researched by Ryan Smith.

Contact

advisories@hustlelabs.com

Details

The code uses a C++ class in order to decompress an LHA archive. The following code segment shows the allocation for the class memory.

```
642B4618 018                    push    7080h
642B461D 01C                    mov     esi, 8
642B4622 01C                    call    ??2@YAPAXI@Z    ; operator new(uint)
```

Subsequently, the program decodes the LHA header. A pointer is passed to the function that is 0x6F48 bytes into the previously depicted class memory. This pointer is the first argument on the stack.

```
642E8780 030                    mov     ecx, [esi+7074h]
642E8786 030                    lea     eax, [esi+6F48h]
642E878C 030                    push    eax
642E878D 034                    push    ecx
642E878E 038                    mov     ecx, esi
642E8790 038                    call    LZA_DecomposeHeader
```

The program then uses EBP as a non-volatile reference to the second argument (arg_4) as shown in the following picture.

```
642E76A0      LZA_DecomposeHeader proc near
642E76A0
642E76A0      var_1430          = dword ptr -1430h
642E76A0      var_142C          = dword ptr -142Ch
642E76A0      var_1428          = dword ptr -1428h
642E76A0      var_1424          = dword ptr -1424h
642E76A0      var_1420          = dword ptr -1420h
642E76A0      var_1418          = dword ptr -1418h
642E76A0      var_1414          = byte ptr -1414h
642E76A0      var_1413          = byte ptr -1413h
642E76A0      var_1412          = byte ptr -1412h
642E76A0      var_1411          = byte ptr -1411h
642E76A0      var_1410          = dword ptr -1410h
642E76A0      var_ulSzDNAME     = dword ptr -140Ch
642E76A0      var_1408          = byte ptr -1408h
642E76A0      var_1407          = byte ptr -1407h
642E76A0      var_DNAME         = byte ptr -1404h
642E76A0      var_1004          = byte ptr -1004h
642E76A0      var_24            = dword ptr -24h
642E76A0      var_1C            = dword ptr -1Ch
642E76A0      var_14            = dword ptr -14h
642E76A0      var_4             = dword ptr -4
642E76A0      arg_0             = dword ptr  4
642E76A0      arg_4             = dword ptr  8
642E76A0
642E76A0 000                    mov     eax, 1410h
642E76A5 000                    call    __alloca_probe
642E76AA 1410                   mov     eax, dword_643695B0
642E76AF 1410                   push    ebx
642E76B0 1414                   mov     ebx, ds:?readin@CGeneri
642E76B6 1414                   push    ebp
642E76B7 1418                   mov     ebp, [esp+1418h+arg_4]
```

Next, the program begins to process an LHA extended-header. Where appropriate, the program reads in extra data from the file. Then, depending on the type of extended-header the program is processing, a switch statement is executed.

```
642E7A00      DIGEST_HDR_LOOP_HEAD:                   ; CODE XREF: LZA_Dec
642E7A00 1420                   cmp     [ebp+LhaHdrInt.bHdrType], 2
642E7A04 1420                   jz      short loc_642E7A38
642E7A06 1420                   mov     eax, [esi+2D04h]
642E7A0C 1420                   lea     edx, [esp+1420h+var_4]
642E7A13 1420                   sub     edx, eax
642E7A15 1420                   cmp     edx, ecx
642E7A17 1420                   jl      diss0
642E7A1D 1420                   push    ecx
642E7A1E 1424                   mov     ecx, [esp+1424h+arg_0]
642E7A25 1424                   push    eax
642E7A26 1428                   call    ds:?readin@CGenericFile@@QAE_NPAXI@Z
642E7A2C 1420                   test    al, al
642E7A2E 1420                   jz      diss0
642E7A34 1420                   mov     ecx, [esp+1420h+var_1410]
642E7A38
642E7A38      loc_642E7A38:                           ; CODE XREF: LZA_Dec
642E7A38 1420                   mov     eax, [esi+2D04h]
642E7A3E 1420                   movzx   edx, byte ptr [eax]
642E7A41 1420                   inc     eax
642E7A42 1420                   cmp     edx, 54h
642E7A45 1420                   mov     [esi+2D04h], eax
642E7A4B 1420                   ja      LHAHDR_HNDLR_UNKNWN
642E7A51 1420                   movzx   edx, ds:LHA_HEADER_NORMALIZE[edx]
642E7A58 1420                   jmp     ds:LHA_HEADER_HANDLER[edx*4]
```

Switch case 1, responsible for processing filename extended headers, is shown in the picture below. The program correctly guards against any size that is greater-than 0xFF before copying the data to the field_14 buffer.

```
642E7A6E      LHAHDR_HNDLR_FNAME:                     ; CODE XREF: LZ
642E7A6E                                              ; DATA XREF: .t
642E7A6E 1420                 lea     edx, [ecx-3]
642E7A71 1420                 mov     ebx, edx
642E7A73 1420                 cmp     ebx, 100h
642E7A79 1420                 jb      short loc_642E7A80
642E7A7B 1420                 mov     ebx, 0FFh
642E7A80
642E7A80          loc_642E7A80:                       ; CODE XREF: LZ
642E7A80 1420                 xor     ecx, ecx
642E7A82 1420                 test    ebx, ebx
642E7A84 1420                 jle     short loc_642E7AA8
642E7A86 1420                 jmp     short loc_642E7A90
642E7A86                  ; ---------------------------------------------------------------------------
642E7A88 1420                 align 10h
642E7A90
642E7A90          loc_642E7A90:                       ; CODE XREF: LZ
642E7A90                                              ; LZA_Decompose
642E7A90 1420                 mov     al, [eax]
642E7A92 1420                 mov     [ecx+ebp+LhaHdrInt.field_14], a
642E7A96 1420                 mov     eax, [esi+2D04h]
642E7A9C 1420                 inc     eax
642E7A9D 1420                 inc     ecx
642E7A9E 1420                 cmp     ecx, ebx
642E7AA0 1420                 mov     [esi+2D04h], eax
642E7AA6 1420                 jl      short loc_642E7A90
642E7AA8
642E7AA8          loc_642E7AA8:                       ; CODE XREF: LZ
642E7AA8 1420                 cmp     ebx, edx
642E7AAA 1420                 jge     short loc_642E7AB6
642E7AAC 1420                 sub     eax, ebx
642E7AAE 1420                 add     eax, edx
642E7AB0 1420                 mov     [esi+2D04h], eax
642E7AB6
642E7AB6          loc_642E7AB6:                       ; CODE XREF: LZ
642E7AB6 1420                 mov     [ebx+ebp+LhaHdrInt.field_14], 0
642E7ABB 1420                 jmp     LHAHDR_GET_NEXT_HDR
```

The following is switch case 2, responsible for processing extended-header directory names. This code ensures that no more than 0x3FF bytes are copied to the var_DNAME buffer; a correct course of action.

```
642E7AC0      LHAHDR_HNDLR_DNAME:                     ; CODE XREF: LZA_D
642E7AC0                                              ; DATA XREF: .text
642E7AC0 1420                 lea     edi, [ecx-3]
642E7AC3 1420                 mov     edx, edi
642E7AC5 1420                 cmp     edx, 400h
642E7ACB 1420                 mov     [esp+1420h+var_ulSzDNAME], edx
642E7ACF 1420                 jb      short loc_642E7ADA
642E7AD1 1420                 mov     edx, 3FFh
642E7AD6 1420                 mov     [esp+1420h+var_ulSzDNAME], edx
642E7ADA
642E7ADA          loc_642E7ADA:                       ; CODE XREF: LZA_D
642E7ADA 1420                 xor     ecx, ecx
642E7ADC 1420                 test    edx, edx
642E7ADE 1420                 jle     short loc_642E7AF6
642E7AE0
642E7AE0          loc_642E7AE0:                       ; CODE XREF: LZA_D
642E7AE0 1420                 mov     dl, [eax]
642E7AE2 1420                 inc     eax
642E7AE3 1420                 mov     [esp+ecx+1420h+var_DNAME], dl
642E7AE7 1420                 mov     edx, [esp+1420h+var_ulSzDNAME]
642E7AEB 1420                 inc     ecx
642E7AEC 1420                 cmp     ecx, edx
642E7AEE 1420                 mov     [esi+2D04h], eax
642E7AF4 1420                 jl      short loc_642E7AE0
642E7AF6
642E7AF6          loc_642E7AF6:                       ; CODE XREF: LZA_D
642E7AF6 1420                 mov     ecx, [esp+1420h+var_ulSzDNAME]
642E7AFA 1420                 cmp     ecx, edi
642E7AFC 1420                 jge     short loc_642E7B08
642E7AFE 1420                 sub     eax, ecx
642E7B00 1420                 add     eax, edi
642E7B02 1420                 mov     [esi+2D04h], eax
642E7B08
642E7B08          loc_642E7B08:                       ; CODE XREF: LZA_D
642E7B08 1420                 mov     eax, [esp+1420h+var_ulSzDNAME]
642E7B0C 1420                 push    '/'
642E7B0E 1424                 lea     ecx, [esp+1424h+var_DNAME]
642E7B12 1424                 push    ecx
642E7B13 1428                 mov     ecx, esi
642E7B15 1428                 mov     [esp+eax+1428h+var_DNAME], 0
642E7B1A 1428                 call    replace_bad_chars
642E7B1F 1420                 jmp     LHAHDR_GET_NEXT_HDR
```

The following code is executed after the program has processed a full LHA header including the extended headers. This code concatenates two user-supplied buffers, the field_14 buffer, and the var_DNAME buffer. The previous code shows that the resulting concatenation may be up to 0x4FE (0x3FF+0xFF+1) bytes in length.

```
642E7CCA 1420                lea     edx, [ebp+LhaHdrInt.field_14]
642E7CCD 1420                mov     eax, edx
642E7CCF 1420                mov     esi, edx
642E7CD1
642E7CD1     loc_642E7CD1:                           ; CODE XREF: LZA_De
642E7CD1 1420                mov     cl, [eax]
642E7CD3 1420                inc     eax
642E7CD4 1420                test    cl, cl
642E7CD6 1420                jnz     short loc_642E7CD1
642E7CD8 1420                lea     edi, [esp+1420h+var_DNAME]
642E7CDC 1420                sub     eax, esi
642E7CDE 1420                dec     edi
642E7CDF 1420                nop
642E7CE0
642E7CE0     loc_642E7CE0:                           ; CODE XREF: LZA_De
642E7CE0 1420                mov     cl, [edi+1]
642E7CE3 1420                inc     edi
642E7CE4 1420                test    cl, cl
642E7CE6 1420                jnz     short loc_642E7CE0
642E7CE8 1420                mov     ecx, eax
642E7CEA 1420                shr     ecx, 2
642E7CED 1420                rep movsd
642E7CEF 1420                mov     ecx, eax
642E7CF1 1420                and     ecx, 3
642E7CF4 1420                lea     eax, [esp+1420h+var_DNAME]
642E7CF8 1420                rep movsb
642E7CFA 1420                mov     ecx, eax
642E7CFC 1420                sub     edx, ecx
642E7CFE 1420                mov     edi, edi
642E7D00
642E7D00     loc_642E7D00:                           ; CODE XREF: LZA_De
642E7D00 1420                mov     cl, [eax]
642E7D02 1420                mov     [eax+edx], cl
642E7D05 1420                inc     eax
642E7D06 1420                test    cl, cl
642E7D08 1420                jnz     short loc_642E7D00
```

The heap buffer's is 0x7080 bytes in size. Since we're writing into this buffer at an offset of 0x6F48, with a size of 0x4FE bytes, we can write 0x3C6 bytes past the end of the allocated memory. This vulnerability allows a standard windows heap attack or a vtable-overwrite attack to be carried out.

Remediation

The code should be modified to either truncate the file and pathname combination, or to enlarge the buffer that holds the result.

Avast anti-virus Version 4.7.869 resolves this issue for the Desktop

Timeline of Events

04-July-2006 – Advisory drafted
11-July-2006 – Vendor notification
14-July-2006 – Vendor created the patch
06-August-2006 – Vendor released the patched version for the Desktop
25-August-2006 – Notification date missed due to Vendor issue
07-September-2006 – Vendor released the patched version for the Server

Attributions

Code and cross-reference screenshots captured using IDA (http://www.datarescue.com).

Flawed code obtained from alwil software (http://www.avast.com).

The images of red mackerel tabby cats were taken from Wikipedia (http://www.wikipedia.org) and the Frederick County Animal Control Center's website (http://www.pethelp.net).

The Creative Commons license-notification image borrowed from http://www.creativecommons.org.

License