Talos Vulnerability Report

TALOS-2021-1291

AT&T Labs Xmill XML decompression DecodeTreeBlock multiple heap-based buffer overflow vulnerabilities

AUGUST 10, 2021

CVE NUMBER

CVE-2021-21826, CVE-2021-21827, CVE-2021-21828

Summary

Multiple heap-based buffer overflow vulnerabilities exists in the XML Decompression DecodeTreeBlock functionality of AT&T Labs Xmill 0.7. A specially crafted XMI File can lead to remote code execution. An attacker can provide a malicious file to trigger these vulnerabilities.

Tested Versions

AT&T Labs Xmill 0.7
Schneider Electric EcoStruxure Control Expert 15

Product URLs

None

CVSSv3 Score

8.1 - CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

CWE

CWE-120 - Buffer Copy without Checking Size of Input (‘Classic Buffer Overflow’)

Details

Xmill and Xdemill are utilities that are purpose built for XML compressed and decompression respectively. These utilities claim to be roughly two times more efficient at compressing XML than other compression methods.

While this software is old, released in 1999, it can be found in modern software suites such as Schneider Electric’s EcoStruxure Control Expert.

Xdemill is used to decompress XMI files compressed by Xmill. This is an XML specific compression program. Both compression and decompression share a combined code base and compilation flags are used to determine the feature set for the output binary. The `Uncompress` function is the main decompression function called directly from `main`

```
void Uncompress(char *sourcefile,char *destfile)
   // The main compres function
{
   ...

   unsigned long blockidx=0;

   mainmem.StartNewMemBlock();

   while(UncompressBlockHeader(&input)==0)
   {
      compressman.UncompressLargeGlobalData(&input);
      uncomprcont.UncompressLargeContainers(&input);

      uncomprcont.Init();

      uncomprtreecont      =uncomprcont.GetContBlock(0)->GetContainer(0);
      uncomprwhitespacecont=uncomprcont.GetContBlock(0)->GetContainer(1);
      uncomprspecialcont   =uncomprcont.GetContBlock(0)->GetContainer(2);
#ifdef TIMING
      c2=clock();
#endif

      DecodeTreeBlock(uncomprtreecont,uncomprwhitespacecont,uncomprspecialcont,&output);
      ...
```

Within `DecodeTreeBlock` there are multiple vulnerabilities that allow for remote code execution via heap buffer overflows.

CVE-2021-21826 - TREETOKEN_WHITESPACE heap buffer overflow

Within `DecodeTreeBlock` which is called during the decompression of an XMI file, a UINT32 is loaded from the file and used as trusted input as the length of a buffer.

```
void DecodeTreeBlock(UncompressContainer *treecont,UncompressContainer *whitespacecont,UncompressContainer *specialcont,XMLOutput *output)
{
   char              *strptr;
   unsigned char     isattrib;
   int               mystrlen;
   static char       tmpstr[20];

   unsigned char     *curptr,*endptr;
   long              id;
   char              isneg;

   curptr=treecont->GetDataPtr();
   endptr=curptr+treecont->GetSize();

   while(curptr<endptr)
   {
      id=LoadSInt32(curptr,&isneg);

      if(isneg==0)   // Do we have a label ID ?
      {
         ...

         switch(id)
         {
         ...

         case TREETOKEN_WHITESPACE:  // A white-space token ?
            mystrlen=whitespacecont->LoadUInt32();
            output->whitespaces((char *)whitespacecont->GetDataPtr(mystrlen),mystrlen);
            break;

        ...
```

This length is then passed to a `memcpy` without verification that the buffer is large enough, or the data is long enough to safely copy.

```
void OUTPUT_STATIC whitespaces(char *str,int len)
{
   characters(str,len);
}

void OUTPUT_STATIC characters(char *str,int len)
{
   switch(x.status)
   {
   case XMLOUTPUT_OPENATTRIB:
      StoreData(str,len);
      return;

   case XMLOUTPUT_OPENLABEL:
      StoreChar('>');

   case XMLOUTPUT_AFTERDATA:
   case XMLOUTPUT_AFTERENDLABEL:
   case XMLOUTPUT_INIT:
      StoreData(str,len);
   }
   x.status=XMLOUTPUT_AFTERDATA;
}

void OUTPUT_STATIC StoreData(char *ptr,int len)
   // Stores the data at position 'ptr' of length 'len'
{
   while(bufsize-curpos<len)
   {
      mymemcpy(buf+curpos,ptr,bufsize-curpos);
      len-=bufsize-curpos;
      ptr+=bufsize-curpos;
      curpos=bufsize;
      Flush();
   }
   mymemcpy(buf+curpos,ptr,len);
   curpos+=len;
}
```

This unsafe `memcpy` results in a controllable heap buffer overflow.

Crash Information

```
Program received signal SIGSEGV, Segmentation fault.
__memcpy_ssse3 () at ../sysdeps/i386/i686/multiarch/memcpy-ssse3.S:1148
1148    ../sysdeps/i386/i686/multiarch/memcpy-ssse3.S: No such file or directory.
 [3;J
[ Legend: Modified register | Code | Heap | Stack | String ]
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
─────────────────── registers ────
$eax   : 0x080ecfc8  →  0x00000000
$ebx   : 0xb7dd1000  →  0x001b2db0
$ecx   : 0xdb78
$edx   : 0x0808afc0  →  0x00000000
$esp   : 0xbffeec44  →  0x080eab80  →  0x00000000
$ebp   : 0x10000
$esi   : 0x17010dce
$edi   : 0x8
$eip   : 0xb7d4692a  →  <__memcpy_ssse3+3242> movaps xmm5, XMMWORD PTR [eax+0x38]
$eflags: [carry PARITY adjust zero sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0033
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────── stack ────
0xbffeec44│+0x0000: 0x080eab80  →  0x00000000     ← $esp
0xbffeec48│+0x0004: 0x08082464  →  0xb7dd1d60  →  0xfbad2a84
0xbffeec4c│+0x0008: 0x08067990  →  <Output::StoreData(char*,+0> mov ecx, DWORD PTR [ebx+0xc]
0xbffeec50│+0x000c: 0x08088b78  →  0x00000000
0xbffeec54│+0x0010: 0x080eab80  →  0x00000000
0xbffeec58│+0x0014: 0x00010000
0xbffeec5c│+0x0018: 0xb7dd1d60  →  0xfbad2a84
0xbffeec60│+0x001c: 0x0808992c  →  0x00000000
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
───────────────── code:x86:32 ────
   0xb7d4691e <__memcpy_ssse3+3230> movaps xmm2, XMMWORD PTR [eax+0x8]
   0xb7d46922 <__memcpy_ssse3+3234> movaps xmm3, XMMWORD PTR [eax+0x18]
   0xb7d46926 <__memcpy_ssse3+3238> movaps xmm4, XMMWORD PTR [eax+0x28]
 → 0xb7d4692a <__memcpy_ssse3+3242> movaps xmm5, XMMWORD PTR [eax+0x38]
   0xb7d4692e <__memcpy_ssse3+3246> movaps xmm7, xmm5
   0xb7d46931 <__memcpy_ssse3+3249> palignr xmm5, xmm4, 0x8
   0xb7d46937 <__memcpy_ssse3+3255> palignr xmm4, xmm3, 0x8
   0xb7d4693d <__memcpy_ssse3+3261> movaps XMMWORD PTR [edx+0x30], xmm5
   0xb7d46941 <__memcpy_ssse3+3265> palignr xmm3, xmm2, 0x8
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
───────────────────── threads ────
[#0] Id 1, Name: "xdemill", stopped 0xb7d4692a in __memcpy_ssse3 (), reason: SIGSEGV
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────── trace ────
[#0] 0xb7d4692a → __memcpy_ssse3()
[#1] 0x8082464 → pathexprman()
[#2] 0x8067990 → Output::StoreData(this=<optimized out>, ptr=<optimized out>, len=<optimized out>)
[#3] 0x8066350 → XMLOutput::characters(this=<optimized out>, str=<optimized out>, len=<optimized out>)
[#4] 0x8065a7a → XMLOutput::whitespaces(this=0x8082464 <output>, str=0x80ecfc8 "", len=<optimized out>)
[#5] 0x8065a7a → DecodeTreeBlock(treecont=<optimized out>, whitespacecont=<optimized out>, specialcont=<optimized out>, output=<optimized
out>)
[#6] 0x8071656 → Uncompress(sourcefile=<optimized out>, destfile=<optimized out>)
[#7] 0x8070ba8 → HandleSingleFile(file=0xbfffed4c "fut.xmi")
[#8] 0x80721ce → HandleFileArg(filepattern=<optimized out>)
[#9] 0x80721ce → main(argc=<optimized out>, argv=<optimized out>)
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────

gef➤
```

CVE-2021-21827 - TREETOKEN_ATTRIBWHITESPACE heap buffer overflow

Within `DecodeTreeBlock` which is called during the decompression of an XMI file, a UINT32 is loaded from the file and used as trusted input as the length of a buffer.

```
void DecodeTreeBlock(UncompressContainer *treecont,UncompressContainer *whitespacecont,UncompressContainer *specialcont,XMLOutput *output)
{
   char              *strptr;
   unsigned char     isattrib;
   int               mystrlen;
   static char       tmpstr[20];

   unsigned char     *curptr,*endptr;
   long              id;
   char              isneg;

   curptr=treecont->GetDataPtr();
   endptr=curptr+treecont->GetSize();

   while(curptr<endptr)
   {
      id=LoadSInt32(curptr,&isneg);

      if(isneg==0)   // Do we have a label ID ?
      {
         ...

         switch(id)
         {
         ...

         case TREETOKEN_ATTRIBWHITESPACE:  // A attrib white-space token ?
            mystrlen=whitespacecont->LoadUInt32();
            output->attribWhitespaces((char *)whitespacecont->GetDataPtr(mystrlen),mystrlen);
            break;

        ...
```

This length is then passed to a `memcpy` without verification that the buffer is large enough, or the data is long enough to safely copy.

```
void OUTPUT_STATIC attribWhitespaces(char *str,int len)
{
   char *ptr=GetDataPtr(len);
   mymemcpy(ptr,str,len);
   x.attribwhitespace=1;
}
```

This results in a heap buffer overflow.

Crash Information

```
Program received signal SIGSEGV, Segmentation fault.
0x0025001b in ?? ()
 [3;J
[ Legend: Modified register | Code | Heap | Stack | String ]
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────── registers ────
$eax   : 0x08082330  →  0x0025001b
$ebx   : 0x0807d540  →  0x08082330  →  0x0025001b
$ecx   : 0xbffee9f0  →  0x080a8ff0  →  0x08082728  →  0x00000000
$edx   : 0x0807d540  →  0x08082330  →  0x0025001b
$esp   : 0xbffee97c  →  0xb7eb6c74  →   add esp, 0x10
$ebp   : 0x1
$esi   : 0x0807d540  →  0x08082330  →  0x0025001b
$edi   : 0xbffee9f0  →  0x080a8ff0  →  0x08082728  →  0x00000000
$eip   : 0x25001b
$eflags: [carry PARITY adjust ZERO sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0073 $ss: 0x007b $ds: 0x007b $es: 0x007b $fs: 0x0000 $gs: 0x0033
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────── stack ────
0xbffee97c│+0x0000: 0xb7eb6c74  →   add esp, 0x10     ← $esp
0xbffee980│+0x0004: 0x0807d540  →  0x08082330  →  0x0025001b
0xbffee984│+0x0008: 0xb7ff0010  →  <_dl_runtime_resolve+16> pop edx
0xbffee988│+0x000c: 0xb7eb68e9  →   add ebx, 0x103717
0xbffee98c│+0x0010: 0x08070ba7  →  <HandleSingleFile(char*)+1031> add cl, ch
0xbffee990│+0x0014: 0x00000003
0xbffee994│+0x0018: 0xbffee9f8  →  0x00000252
0xbffee998│+0x001c: 0x080a8ff0  →  0x08082728  →  0x00000000
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────── code:x86:32 ────
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x25001b
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────── threads ────
[#0] Id 1, Name: "xdemill", stopped 0x25001b in ?? (), reason: SIGSEGV
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────── trace ────
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────
gef➤
```

CVE-2021-21828 - Default case global buffer overflow

In the default case of `DecodeTreeBlock` a label is created via `CurPath::AddLabel` in order to track the label for later reference.

```
void DecodeTreeBlock(UncompressContainer *treecont,UncompressContainer *whitespacecont,UncompressContainer *specialcont,XMLOutput *output)
{
   char              *strptr;
   unsigned char     isattrib;
   int               mystrlen;
   static char       tmpstr[20];

   unsigned char     *curptr,*endptr;
   long              id;
   char              isneg;

   curptr=treecont->GetDataPtr();
   endptr=curptr+treecont->GetSize();

   while(curptr<endptr)
   {
      id=LoadSInt32(curptr,&isneg);

      if(isneg==0)   // Do we have a label ID ?
      {
         if(id>=32768L)
         {
            Error("Error while decompressing file!");
            Exit();
         }

         switch(id)
         {
         ...

         default: // Do we have a start label token?
            id-=LABELIDX_TOKENOFFS;
            mystrlen=globallabeldict.LookupLabel((TLabelID)id,&strptr,&isattrib);

            if(isattrib==0)
               output->startElement(strptr,mystrlen);
            else
               output->startAttribute(strptr,mystrlen);

            curpath.AddLabel((TLabelID)id);
         }
      }
      else  // We have a block ID ==> I.e. we have some text
         uncomprcont.GetContBlock(id)->UncompressText(output);
   }
}
```

Within `CurPath::AddLabel` a global pointer is incremented to store the various `TLabelID` for reference later in the decompression process, this pointer is unchecked and can be used to write 2 byte chunks to a global buffer region of memory which can result in control of the instruction pointer.

```
void AddLabel(TLabelID labelid)
      // Add a label at the end of the path
   {
      ...

      if(curlabel==curblock->labels+CURPATH_LABELBLOCKSIZE)
         // Is there not enough space in the current block?
         // ==> Create new block, if there is no next block
         // (We never delete blocks)
      {
         if(curblock->next==NULL)
         {
            curblock->next=new CurPathLabelBlock();
            curblock->next->prev=curblock;
            curblock->next->next=NULL;
         }
         curblock=curblock->next;

         // We set the new current label
         curlabel=curblock->labels;
      }
      *curlabel=labelid;
      curlabel++;
   }
```

Crash Information

```
=================================================================
==32266==ERROR: AddressSanitizer: global-buffer-overflow on address 0x08691a58 at pc 0x08186ebd bp 0xbfa35cc8 sp 0xbfa35cbc
WRITE of size 2 at 0x08691a58 thread T0
    #0 0x8186ebc in CurPath::AddLabel(unsigned short) /home/fuzz/Desktop/xmill/./src/CurPath.hpp:139:16
    #1 0x81840bc in DecodeTreeBlock(UncompressContainer*, UncompressContainer*, UncompressContainer*, XMLOutput*)
/home/fuzz/Desktop/xmill/./src/Decode.cpp:78:13
    #2 0x8197226 in Uncompress(char*, char*) /home/fuzz/Desktop/xmill/./src/Main.cpp:854:7
    #3 0x8196c37 in HandleSingleFile(char*) /home/fuzz/Desktop/xmill/./src/Main.cpp:248:10
    #4 0x8197482 in HandleFileArg(char*) /home/fuzz/Desktop/xmill/./src/Main.cpp:382:4
    #5 0x81976f5 in main /home/fuzz/Desktop/xmill/./src/Main.cpp:494:7
    #6 0xb7bc6646 in __libc_start_main /build/glibc-ViVLyQ/glibc-2.23/csu/../csu/libc-start.c:291
    #7 0x80664d3 in _start (/home/fuzz/Desktop/xmill/unix/xdemill+0x80664d3)

0x08691a58 is located 8 bytes to the left of global variable 'curpath' defined in './src/Main.cpp:69:19' (0x8691a60) of size 80
0x08691a58 is located 48 bytes to the right of global variable 'uncomprcont' defined in './src/UnCompCont.cpp:40:25' (0x8691a20) of size 8
SUMMARY: AddressSanitizer: global-buffer-overflow /home/fuzz/Desktop/xmill/./src/CurPath.hpp:139:16 in CurPath::AddLabel(unsigned short)
Shadow bytes around the buggy address:
  0x210d22f0: 00 00 04 f9 f9 f9 f9 f9 00 f9 f9 f9 f9 f9 f9 f9
  0x210d2300: 00 00 f9 f9 f9 f9 f9 f9 00 00 f9 f9 f9 f9 f9 f9
  0x210d2310: 00 00 f9 f9 f9 f9 f9 f9 00 00 04 f9 f9 f9 f9 f9
  0x210d2320: 00 00 04 f9 f9 f9 f9 f9 00 00 04 f9 f9 f9 f9 f9
  0x210d2330: 00 00 04 f9 f9 f9 f9 f9 00 f9 f9 f9 f9 f9 f9 f9
=>0x210d2340: 00 00 00 00 00 f9 f9 f9 f9 f9 f9[f9]00 00 00 00
  0x210d2350: 00 00 00 00 00 00 f9 f9 f9 f9 f9 f9 00 04 f9 f9
  0x210d2360: f9 f9 f9 f9 00 04 f9 f9 f9 f9 f9 f9 00 00 00 00
  0x210d2370: 04 f9 f9 f9 f9 f9 f9 f9 04 f9 f9 f9 f9 f9 f9 f9
  0x210d2380: 04 f9 f9 f9 f9 f9 f9 f9 04 f9 f9 f9 f9 f9 f9 f9
  0x210d2390: 00 00 04 f9 f9 f9 f9 f9 00 00 04 f9 f9 f9 f9 f9
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
Error in file '../fuzz/triage/fut.xmi':
Error while decompressing file!
```

Timeline

2021-05-03 - Vendor Disclosure

2021-08-10 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

NEXT REPORT

TALOS-2021-1292

TALOS-2021-1290