Talos Vulnerability Report

TALOS-2021-1278

AT&T Labs Xmill XML parsing ParseAttribs memory corruption vulnerability

AUGUST 10, 2021

CVE NUMBER

CVE-2021-21810

Summary

A memory corruption vulnerability exists in the XML-parsing ParseAttribs functionality of AT&T Labs' Xmill 0.7. A specially crafted XML file can lead to a heap buffer overflow. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

AT&T Labs Xmill 0.7

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-122 - Heap-based Buffer Overflow

Details

Xmill and Xdemill are utilities that are purpose-built for XML compression 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.

Within the XML Parsing functionality of Xmill, if an attribute is encoutered without a subsequent `=` and the `curblock` is set to 0x0 a heap buffer overflow will occur, overwriting arbitrary heap memory 2 bytes at a time.

To first overwrite `curblock` to NULL, the first start level and end label cannot match.

```
void SAXClient::HandleEndLabel(char *str,int len,char iscont)
   // Stores the end label
{
   TLabelID labelid=curpath.RemoveLabel();
   TLabelID endlabelid;

   ...

   // Was the current path empty? I.e. we didn't have any corresponding starting label?
   // ==> Exit
   if(labelid==LABEL_UNDEFINED)
   {
      ...
   }

   if(str==NULL)  // Did we have an empty element of the form <label/> ?
      ...
   else
   {
      // Otherwise, let's check whether the end label is the same as the start label
      endlabelid=globallabeldict.FindLabelOrAttrib(str,len,0);

      if(endlabelid!=labelid) // Not the same?
                              // We look at the previous label in the path
                              // If this is not equal either, then we exit
      {
         char *ptr;
         unsigned long startlen=globallabeldict.LookupCompressLabel(labelid,&ptr);

         TLabelID prevlabelid=curpath.RemoveLabel();

         ...
      }
      ...
   }
}
```

In `HandleEndLabel` the labels are stored on a stack of labels. With only a single StartLabel in existence, the first call to `RemoveLabel` will pop it off the stack. The second call to `RemoveLabel` during the error handling of mismatched labels, causes a NULL to written to `curblock`.

```
TLabelID RemoveLabel()
    // Removes the last label from the stack
{
....

    curlabel--;
    if(curlabel==curblock->labels-1)
        // If the label in the previou block?
        // Go one block back
    {
        curblock=curblock->prev;
        if(curblock==NULL)   // No previous block? => Exit
        return LABEL_UNDEFINED;

        curlabel=curblock->labels+CURPATH_LABELBLOCKSIZE-1;
    }
    return *curlabel;
}
```

Because the label attempts to move backward a single block to look for previous labels that don't exist, NULL is written to `curblock` it then returns `LABEL_UNDEFINED` which is not checked for in `HandleEndLabel`. At this point `curblock` is set to NULL and parsing continues. Once a new label is found with attributes, `ParseAttribs` will be called.

```
char ParseAttribs()
// This function scans the attributes in a given start label
// The returns as soon as the trailing '>' is reached
{
    char c;
    char *strptr;
    int  len;

    do
    {
        while(ReadWhiteSpaces(&strptr,&len)==0)
            // We read all white-spaces
            saxclient->HandleAttribWhiteSpaces(strptr,len,1);

        saxclient->HandleAttribWhiteSpaces(strptr,len,0);

        // Now we don't have any more white-spaces and we search
        // for '=' (if there is an attribute) or '>' (for the end of the element)
        PeekChar(&c);
        if((c=='>')||(c=='/'))  // End of label?
        {
            SkipChar();
            return c;
        }
        // Let's find '=' or some white-space
        while(ReadStringUntil(&strptr,&len,1,'=',0)==0)
            // We scan until we reach '='
            saxclient->HandleAttribName(strptr,len,1);

        // We found '='
        saxclient->HandleAttribName(strptr,len-1,0);
        ...
```

If an attribute is found `ParseAttribs` will be called to parse it. If this attribute does not contain an `=` then `ParseAttribs` will continuously call `HandleAttribName` due to `ReadStringUntil` always returning 0.

```
void SAXClient::HandleAttribName(char *str,int len,char iscont)
   // Handles a single attribute name
{
   // We simply create a new attribute, if it does not already exist
   TLabelID labelid=globallabeldict.FindLabelOrAttrib(str,len,1);

   if(labelid==LABEL_UNDEFINED)
      labelid=globallabeldict.CreateLabelOrAttrib(str,len,1);

   // We add it to the current path
   curpath.AddLabel(labelid);

   // We store the label ID
   StoreStartLabel(labelid);
}
```

`HandleAttribName` will try to find the attribute in the `globallabeldict` or create it, but more importantly it calls `AddLabel` every time that `HandleAttribName` is called. Since an `=` is never found, the first call to `HandleAttribName` will create the attribute in the `globallabeldict` but `AddLabel` will continuously be called.

```
void AddLabel(TLabelID labelid)
      // Add a label at the end of the path
   {
#ifdef PROFILE
      curdepth++;
         if(curdepth>maxdepth)
            maxdepth=curdepth;
#endif

      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++;
   }
```

At this point `curblock` is NULL and `curlabel` is a pointer, which turns `AddLabel` into a heap buffer overflow while `*curlabel` is continuously written to and incremented without ever triggering the allocation of a new block. This will result in an out of bounds write of two bytes at a time, resulting in heap corruption.

Crash Information

```
=================================================================
==2676793==ERROR: AddressSanitizer: global-buffer-overflow on address 0x000000f36fe0 at pc 0x000000517c76 bp 0x7fffffecfc0 sp
0x7fffffecfb8
WRITE of size 2 at 0x000000f36fe0 thread T0
    #0 0x517c75 in CurPath::AddLabel(unsigned short) (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x517c75)
    #1 0x515a42 in SAXClient::HandleAttribName(char*, int, char) (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x515a42)
    #2 0x52b001 in XMLParse::ParseAttribs() (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x52b001)
    #3 0x527181 in XMLParse::ParseLabel() (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x527181)
    #4 0x52364c in XMLParse::DoParsing(SAXClient*) (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x52364c)
    #5 0x52161e in Compress(char*, char*) (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x52161e)
    #6 0x5211a6 in HandleSingleFile(char*) (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x5211a6)
    #7 0x521b91 in HandleFileArg(char*) (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x521b91)
    #8 0x522059 in main (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x522059)
    #9 0x7ffff7a71cb1 in __libc_start_main csu/../csu/libc-start.c:314:16
    #10 0x41c84d in _start (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x41c84d)

0x000000f36fe0 is located 0 bytes to the right of global variable 'pathtree' defined in './src/Main.cpp:74:22' (0xeb6fc0) of size 524320
SUMMARY: AddressSanitizer: global-buffer-overflow (/home/fuzz/Desktop/xmill/unix/xmill-asan+0x517c75) in CurPath::AddLabel(unsigned short)
Shadow bytes around the buggy address:
  0x0000801deda0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000801dedb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000801dedc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000801dedd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0000801dede0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0000801dedf0: 00 00 00 00 00 00 00 00 00 00 00 00 00[f9]f9 f9 f9
  0x0000801dee00: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x0000801dee10: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x0000801dee20: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x0000801dee30: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x0000801dee40: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 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
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  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
  Shadow gap:              cc
==2676793==ABORTING
```

Timeline

2021-04-30 - Vendor Disclosure
2021-08-10 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.
