Talos Vulnerability Report

TALOS-2021-1280

AT&T Labs Xmill multiple command line parsing vulnerabilities

AUGUST 10, 2021

CVE NUMBER

CVE-2021-21812, CVE-2021-21813, CVE-2021-21814, CVE-2021-21815

Summary

Multiple stack-based buffer overflow vulnerabilities exists in the command-line-parsing HandleFileArg functionality of AT&T Labs' Xmill 0.7. A specially crafted command-line argument can lead to code execution. An attacker can provide malicious input to trigger these vulnerabilities.

Tested Versions

AT&T Labs Xmill 0.7

Product URLs

None

CVSSv3 Score

7.8 - CVSS:3.0/AV:L/AC:L/PR:L/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 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.

Multiple vulnerabilities exist in the command line argument parsing code within Xmill Xdemill. This code is feature gated for Windows and Linux which expose different vulnerabilities based on the operating system the code was compiled for. These vulnerabilities are easily triggered via command line arguements containing file names that don't need to exist.

CVE-2021-21812 - Windows strcpy no metacharacters

Within the function `HandleFileArg` the argument `filepattern` is under control of the user who passes it in from the command line. `filepattern` is passed directly to `strcpy` copying the path provided by the user into a staticly sized buffer without any length checks resulting in a stack-buffer overflow.

```
void HandleFileArg(char *filepattern)
   // Takes a file name argument from the command line
   // and forward the file names to 'HandleSingleFile'
   // In Windows, file patterns with '*' and '?' must be explicitly
   // resolved.
{
   char        fullpath[400];
#ifdef WIN32
   _finddata_t finddata;
   long        handle;
   char        *ptr;
   int         fullpathlen;

   // Let's check if we have any meta characters '*' or '?' ?
   // We don't have them, we go directly to 'HandleSingleFile'

   ptr=filepattern;
   while(*ptr!=0)
   {
      if((*ptr=='*')||(*ptr=='?'))
         break;
      ptr++;
   }

   if(*ptr==0) // We didn't find any metacharacter?
               // The file name gets directly forwarded to HandleSingleFile
   {
      strcpy(fullpath,filepattern); // BUGHERE
      HandleSingleFile(fullpath);
      return;
   }
   ...
```

Crash Information

```
.\xmill.exe -c
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
=================================================================
==324==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x030ffb98 at pc 0x005a5b03 bp 0x030ff9d8 sp 0x030ff9c0WRITE of size 538 at
0x030ffb98 thread T0
    #0 0x5a5b1d in __asan_wrap_strcpy D:\agent\_work\4\s\src\vctools\crt\asan\llvm\compiler-rt\lib\asan\asan_interceptors.cc:432
    #1 0x543559 in HandleFileArg(char *) (F:\Work\ControlExpert\Xmill\tmp\xmill.exe+0x443559)
    #2 0x543c1c in main (F:\Work\ControlExpert\Xmill\tmp\xmill.exe+0x443c1c)
    #3 0x5bd4db in _scrt_common_main_seh D:\agent\_work\4\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:288
    #4 0x7688fa28  (C:\WINDOWS\System32\KERNEL32.DLL+0x6b81fa28)
    #5 0x776e7c7d  (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2e7c7d)
    #6 0x776e7c4d  (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2e7c4d)

Address 0x030ffb98 is located in stack of thread T0 at offset 416 in frame
    #0 0x507faf in ILT+28560(?HandleFileArg@@YAXPAD@Z) (F:\Work\ControlExpert\Xmill\tmp\xmill.exe+0x407faf)

  This frame has 2 object(s):
    [16, 416) 'fullpath'
    [32, 328) 'finddata' <== Memory access at offset 416 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp, SEH and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow D:\agent\_work\4\s\src\vctools\crt\asan\llvm\compiler-rt\lib\asan\asan_interceptors.cc:432
in __asan_wrap_strcpy
Shadow bytes around the buggy address:
  0x3061ff20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x3061ff30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f1
  0x3061ff40: f1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x3061ff50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x3061ff60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x3061ff70: 00 00 00[f2]f2 f2 f2 00 00 00 00 00 00 00 00 00
  0x3061ff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x3061ff90: 00 00 00 00 00 00 00 00 00 00 00 00 f3 f3 f3 f3
  0x3061ffa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x3061ffb0: 00 00 00 00 00 00 00 00 00 f1 f1 04 f3 f3 f3 f3
  0x3061ffc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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
```

CVE-2021-21813 - Windows memcpy

Within the function `HandleFileArg` the argument `filepattern` is under control of the user who passes it in from the command line. `filepattern` is passed directly to `memcpy` copying the path provided by the user into a staticly sized buffer without any length checks resulting in a stack-buffer overflow.

```
void HandleFileArg(char *filepattern)
   // Takes a file name argument from the command line
   // and forward the file names to 'HandleSingleFile'
   // In Windows, file patterns with '*' and '?' must be explicitly
   // resolved.
{
   char        fullpath[400];
#ifdef WIN32
   _finddata_t finddata;
   long        handle;
   char        *ptr;
   int         fullpathlen;

   // Let's check if we have any meta characters '*' or '?' ?
   // We don't have them, we go directly to 'HandleSingleFile'

   ptr=filepattern;
   while(*ptr!=0)
   {
      if((*ptr=='*')||(*ptr=='?'))
         break;
      ptr++;
   }

   if(*ptr==0) // We didn't find any metacharacter?
               // The file name gets directly forwarded to HandleSingleFile
   {
      ...
   }
   // Otherwise, we apply functions '_findfirst' and '_findnext'

   // We scan from the back of the file name and look
   // for a separator
   ptr=filepattern+strlen(filepattern)-1;

   while(ptr>=filepattern)
   {
      if((*ptr=='\\')||(*ptr=='/'))
         break;
      ptr--;
   }

   if(ptr<filepattern)   // We didn't find a separator ?
   {
      // The file path is empty
      *fullpath=0;
      fullpathlen=0;
   }
   else
   {
      // We the path part from the file pattern including
      // the separator that we found
      memcpy(fullpath,filepattern,ptr-filepattern+1); // BUGHERE
      fullpath[ptr-filepattern+1]=0;
      fullpathlen=ptr-filepattern+1;
   }
   ....
```

Crash Information

```
.\xmill.exe -c
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA/*AAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA
=================================================================
==3460==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x02dff4b8 at pc 0x005a3cbc bp 0x02dff2f4 sp 0x02dff2e4
WRITE of size 537 at 0x02dff4b8 thread T0
    #0 0x5a3cd6 in __asan_wrap_memcpy D:\agent\_work\4\s\src\vctools\crt\asan\llvm\compiler-
rt\lib\sanitizer_common\sanitizer_common_interceptors.inc:801
    #1 0x54375a in HandleFileArg(char *) (F:\Work\ControlExpert\Xmill\tmp\xmill.exe+0x44375a)
    #2 0x543c1c in main (F:\Work\ControlExpert\Xmill\tmp\xmill.exe+0x443c1c)
    #3 0x5bd4db in _scrt_common_main_seh D:\agent\_work\4\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl:288
    #4 0x7688fa28  (C:\WINDOWS\System32\KERNEL32.DLL+0x6b81fa28)
    #5 0x776e7c7d  (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2e7c7d)
    #6 0x776e7c4d  (C:\WINDOWS\SYSTEM32\ntdll.dll+0x4b2e7c4d)

Address 0x02dff4b8 is located in stack of thread T0 at offset 416 in frame
    #0 0x507faf in ILT+28560(?HandleFileArg@@YAXPAD@Z) (F:\Work\ControlExpert\Xmill\tmp\xmill.exe+0x407faf)

  This frame has 2 object(s):
    [16, 416) 'fullpath'
    [32, 328) 'finddata' <== Memory access at offset 416 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp, SEH and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow D:\agent\_work\4\s\src\vctools\crt\asan\llvm\compiler-
rt\lib\sanitizer_common\sanitizer_common_interceptors.inc:801 in __asan_wrap_memcpy
Shadow bytes around the buggy address:
  0x305bfe40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x305bfe50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x305bfe60: 00 00 00 f1 f1 00 00 00 00 00 00 00 00 00 00 00
  0x305bfe70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x305bfe80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x305bfe90: 00 00 00 00 00 00 00[f2]f2 f2 f2 00 00 00 00 00
  0x305bfea0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x305bfeb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x305bfec0: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
  0x305bfed0: 00 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 04
  0x305bfee0: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
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
==3460==ABORTING
```

CVE-2021-21814 - Windows NULL out of bounds write

Within the function `HandleFileArg` the argument `filepattern` is under control of the user who passes it in from the command line. `filepattern` is passed directly to `strlen` to determine the ending location of the char* passed in by the user, no checks are done to see if the passed in char* is longer than the staticly sized buffer data is `memcpy`‘d into, but after the `memcpy` a null byte is written to what is assumed to be the end of the buffer to terminate the char*, but without length checks, this null write occurs at an arbitrary offset from the buffer. This bug would exist even if the previous `memcpy` buffer overflow was fixed as shown.

```
void HandleFileArg(char *filepattern)
   // Takes a file name argument from the command line
   // and forward the file names to 'HandleSingleFile'
   // In Windows, file patterns with '*' and '?' must be explicitly
   // resolved.
{
   char        fullpath[400];
#ifdef WIN32
   _finddata_t finddata;
   long        handle;
   char        *ptr;
   int         fullpathlen;

   // Let's check if we have any meta characters '*' or '?' ?
   // We don't have them, we go directly to 'HandleSingleFile'

   ptr=filepattern;
   while(*ptr!=0)
   {
      if((*ptr=='*')||(*ptr=='?'))
         break;
      ptr++;
   }

   if(*ptr==0) // We didn't find any metacharacter?
               // The file name gets directly forwarded to HandleSingleFile
   {
      ...
   }
   // Otherwise, we apply functions '_findfirst' and '_findnext'

   // We scan from the back of the file name and look
   // for a separator
   ptr=filepattern+strlen(filepattern)-1;

   while(ptr>=filepattern)
   {
      if((*ptr=='\\')||(*ptr=='/'))
         break;
      ptr--;
   }

   if(ptr<filepattern)   // We didn't find a separator ?
   {
      // The file path is empty
      *fullpath=0;
      fullpathlen=0;
   }
   else
   {
      // We the path part from the file pattern including
      // the separator that we found

      //BUGFIX
      int copy_length = 0;
      if (ptr-filepattern+1 > 400) {
          copy_length = 400;
      } else {
          copy_length = ptr-filepattern+1;
      }
      memcpy(fullpath,filepattern,copy_length);
      // END BUGFIX
      fullpath[ptr-filepattern+1]=0;  // BUGHERE
      fullpathlen=ptr-filepattern+1;
   }
```

CVE-2021-21815 - Linux strcpy

Within the function `HandleFileArg` the argument `filepattern` is under control of the user who passes it in from the command line. `filepattern` is passed directly to `strcpy` copying the path provided by the user into a staticly sized buffer without any length checks resulting in a stack-buffer overflow.

```
void HandleFileArg(char *filepattern)
   // Takes a file name argument from the command line
   // and forward the file names to 'HandleSingleFile'
   // In Windows, file patterns with '*' and '?' must be explicitly
   // resolved.
{
   char        fullpath[400];
#ifdef WIN32
...
#else

   // In UNIX, the file name expansion is done by the shell
   // ==> We only need to look at the specific file
   strcpy(fullpath,filepattern); // BUGHERE
   HandleSingleFile(fullpath);
#endif
}
```

Crash Information

```
./xmill -c
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA
=================================================================
==889467==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fffffffdc90 at pc 0x000000482908 bp 0x7fffffffdad0 sp 0x7fffffffd290
WRITE of size 437 at 0x7fffffffdc90 thread T0
    #0 0x482907 in strcpy (/home/fuzz/Desktop/xmill/unix/xmill+0x482907)
    #1 0x4fe42b in main (/home/fuzz/Desktop/xmill/unix/xmill+0x4fe42b)
    #2 0x7ffff7a71cb1 in __libc_start_main csu/../csu/libc-start.c:314:16
    #3 0x41caed in _start (/home/fuzz/Desktop/xmill/unix/xmill+0x41caed)

Address 0x7fffffffdc90 is located in stack of thread T0 at offset 432 in frame
    #0 0x4fe02f in main (/home/fuzz/Desktop/xmill/unix/xmill+0x4fe02f)

  This frame has 2 object(s):
    [32, 432) 'fullpath.i'
    [496, 504) 'pathptr' <== Memory access at offset 432 partially underflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/home/fuzz/Desktop/xmill/unix/xmill+0x482907) in strcpy
Shadow bytes around the buggy address:
  0x10007fff7b40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7b50: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
  0x10007fff7b60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7b70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7b80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x10007fff7b90: 00 00[f2]f2 f2 f2 f2 f2 f2 f2 f8 f3 f3 f3 f3 f3
  0x10007fff7ba0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7bb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7bc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7bd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7be0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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
==889467==ABORTING
```

Timeline

2021-04-30 - Vendor Disclosure

2021-08-10 - Public Release

CREDIT

Discovered by Carl Hurd of Cisco Talos.
