

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 exist 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 arguments containing file names that don't need to exist.

CVE-2021-21812 - Windows strcpy no metacharacters

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

```
void HandleFileArg(char *filepath)
// 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    filepath[4096];
#ifdef WIN32
    _finddata_t finddata;
    long      handle;
    char      *ptr;
    int       filepathlen;

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

    ptr=filepath;
    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(filepath,filepath); // BUGHERE
        HandleSingleFile(filepath);
        return;
    }
    ...
}
```

Crash Information

CVE-2021-21813 - Windows memcpy

Within the function `HandleFileArg` the argument `filepath` is under control of the user who passes it in from the command line. `filepath` is passed directly to `memcpy` copying the path provided by the user into a statically 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

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

```

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 statically 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

