Talos Vulnerability Report

TALOS-2022-1453

KiCad EDA Gerber Viewer gerber and excellon coordinates parsing stack-based buffer overflow vulnerability

FEBRUARY 16, 2022

CVE NUMBER

CVE-2022-23804,CVE-2022-23803

Summary

Multiple stack-based buffer overflow vulnerabilities exist in the Gerber Viewer gerber and excellon coordinates parsing functionality of KiCad EDA 6.0.1 and master commit de006fc010. A specially-crafted gerber or excellon file can lead to code execution. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

KiCad EDA 6.0.1
KiCad EDA master commit de006fc010

Product URLs

KiCad EDA - https://www.kicad.org/

CVSSv3 Score

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

CWE

CWE-121 - Stack-based Buffer Overflow

Details

KiCad is a cross-platform open-source software for electronics design automation, used to design and simulate electronic hardware. It offers several tools like schematic and symbol editor, PCB and footprint editor, Gerber viewer and others.

KiCad's Gerber Viewer is found in a separate binary called `gerbview` and allows the viewing of Gerber files, Excellon files and Gerber job files, optionally contained in zip archives.

When opening a Gerber file, the method `GERBER_FILE_IMAGE::LoadGerberFile` in `readgerb.cpp` is called:

```
// size of a single line of text from a gerber file.
// warning: some files can have *very long* lines, so the buffer must be large.
#define GERBER_BUFZ 1000000
// A large buffer to store one line
static char lineBuffer[GERBER_BUFZ+1];  // [7]

bool GERBER_FILE_IMAGE::LoadGerberFile( const wxString&amp;amp; aFullFileName )
{
    int      G_command = 0;        // command number for G commands like G04
    int      D_commande = 0;       // command number for D commands like D02
    char*    text;

    ClearMessageList( );
    ResetDefaultValues();

    // Read the gerber file */
    m_Current_File = wxFopen( aFullFileName, wxT( &amp;#34;rt&amp;#34; ) );

    if( m_Current_File == nullptr )
        return false;

    m_FileName = aFullFileName;

    LOCALE_IO toggleIo;

    wxString msg;

    while( true )
    {
        if( fgets( lineBuffer, GERBER_BUFZ, m_Current_File ) == nullptr )  // [1]
            break;

        m_LineNum++;
        text = StrPurge( lineBuffer );

        while( text &amp;amp;&amp;amp; *text )
        {
            switch( *text )
            {
            case &amp;#39; &amp;#39;:
            case &amp;#39;\r&amp;#39;:
            case &amp;#39;\n&amp;#39;:
                text++;
                break;

            case &amp;#39;*&amp;#39;:       // End command
                m_CommandState = END_BLOCK;
                text++;
                break;

            case &amp;#39;M&amp;#39;:       // End file
                m_CommandState = CMD_IDLE;
                while( *text )
                    text++;
                break;

            case &amp;#39;G&amp;#39;:    /* Line type Gxx : command */
                G_command = GCodeNumber( text );
```

```
                Execute_G_Command( text, G_command );
                break;

            case &#39;D&#39;:       /* Line type Dxx : Tool selection (xx &gt; 0) or
                                * command if xx = 0..9 */
                D_commande = DCodeNumber( text );
                Execute_DCODE_Command( text, D_commande );
                break;

            case &#39;X&#39;:
            case &#39;Y&#39;:                    /* Move or draw command */
                m_CurrentPos = ReadXYCoord( text );                         // [2]
                if( *text == &#39;*&#39; )      // command like X12550Y19250*
                {
                    Execute_DCODE_Command( text, m_Last_Pen_Command );
                }
                break;

            case &#39;I&#39;:                                                        //
[3]
            case &#39;J&#39;:       /* Auxiliary Move command */
                m_IJPos = ReadIJCoord( text );

                if( *text == &#39;*&#39; )      // command like X35142Y15945J504*
                {
                    Execute_DCODE_Command( text, m_Last_Pen_Command );
                }
                break;
```

This method takes a gerber file path, opens it, and parses it line-by-line [1]. When a line starting with "X" or "Y" is encountered, the method `GERBER_FILE_IMAGE::ReadXYCoord` is called [2], whereas when the line starts with "I" or "J", the method `GERBER_FILE_IMAGE::ReadIJCoord` is called [3]. In both cases, the current line is passed as parameter. Both methods lead to the same issue; let's detail them individually.
Also note that these same methods can be reached via an Excellon file in a similar way (via `EXCELLON_IMAGE::LoadFile`), as discussed below.

CVE-2022-23803 - ReadXYCoord

```
VECTOR2I GERBER_FILE_IMAGE::ReadXYCoord( char*&amp;amp; Text, bool aExcellonMode )
{
    VECTOR2I pos;
    int      type_coord = 0, current_coord, nbdigits;
    bool     is_float   = false;
    char*    text;
    char     line[256];      // [4]


    if( m_Relative )
        pos.x = pos.y = 0;
    else
        pos = m_CurrentPos;

    if( Text == nullptr )
        return pos;

    text = line;

    while( *Text )
    {
        if( ( *Text == &amp;#39;X&amp;#39; ) || ( *Text == &amp;#39;Y&amp;#39; ) || ( *Text ==
&amp;#39;A&amp;#39; ) )
        {
            type_coord = *Text;
            Text++;
            text     = line;
            nbdigits = 0;

            while( IsNumber( *Text ) )   // [5]
            {
                if( *Text == &amp;#39;.&amp;#39; )  // Force decimal format if reading a
floating point number
                    is_float = true;

                // count digits only (sign and decimal point are not counted)
                if( (*Text &amp;gt;= &amp;#39;0&amp;#39;) &amp;amp;&amp;amp; (*Text &amp;lt;=&amp;#39;9&amp;#39;) )
                    nbdigits++;

                *(text++) = *(Text++);    // [6]
            }

            *text = 0;

            ...

            continue;
        }
        else
        {
            break;
        }
    }

    if( m_Relative )
    {
        pos.x += m_CurrentPos.x;
        pos.y += m_CurrentPos.y;
```

```
    }

    m_CurrentPos = pos;
    return pos;
}
```

At [4] a `line` buffer of size 256 bytes is allocated on the stack. Since the line started with "X" or "Y", the code reaches the while loop at [5], which expects a number inside the line. The loop stores the current character in the `text` buffer (that is, the `line` buffer) [6] and only stops when `Text` does not point to a number anymore. Because `Text` [7] is much larger than `line` and the loop does not check if [6] operates within the `line`'s buffer bounds, the loop could write out of bounds if a large enough line containing numbers is supplied. This is a straightforward stack-based buffer overflow that could lead to code execution. Moreover, `*text` is assigned later in the same method, so corruption could occur slightly later too.

Allowed characters for `IsNumber` are the following:

```
#define IsNumber( x ) ( ( ( (x) &gt;= &#39;0&#39; ) &amp;&amp; ( (x)
&lt;=&#39;9&#39; ) )   \
                        || ( (x) == &#39;-&#39; ) || ( (x) == &#39;+&#39; )  || ( (x)
== &#39;.&#39; ) )
```

Note that this method is also used while parsing an Excellon file. In that case, this same issue can be triggered via `Execute_EXCELLON_G_Command` or `Execute_Drill_Command`, both eventually calling `ReadXYCoord`.

CVE-2022-23804 - ReadIJCoord

```
VECTOR2I GERBER_FILE_IMAGE::ReadIJCoord( char*&amp;amp; Text )
{
    VECTOR2I pos( 0, 0 );

    int     type_coord = 0, current_coord, nbdigits;
    bool    is_float   = false;
    char*   text;
    char    line[256];         // [4]

    if( Text == nullptr )
        return pos;

    text = line;

    while( *Text )
    {
        if( ( *Text == &amp;#39;I&amp;#39; ) || ( *Text == &amp;#39;J&amp;#39; ) )
        {
            type_coord = *Text;
            Text++;
            text     = line;
            nbdigits = 0;

            while( IsNumber( *Text ) )   // [5]
            {
                if( *Text == &amp;#39;.&amp;#39; )
                    is_float = true;

                // count digits only (sign and decimal point are not counted)
                if( ( *Text &amp;gt;= &amp;#39;0&amp;#39; ) &amp;amp;&amp;amp; ( *Text &amp;lt;= &amp;#39;9&amp;#39;
) )
                    nbdigits++;

                *(text++) = *(Text++);   // [6]
            }

            *text = 0;

            ...

            continue;
        }
        else
        {
            break;
        }
    }

    m_IJPos = pos;
    m_LastArcDataType = ARC_INFO_TYPE_CENTER;
    m_LastCoordIsIJPos = true;

    return pos;
}
```

At [4] a `line` buffer of size 256 bytes is allocated on the stack. Since the line started with "I" or "J", the code reaches the while loop at [5], which expects a number inside the line. The loop stores the current character in the `text` buffer (that is, the `line` buffer) [6] and only stops when `Text` does not point to a number anymore.
Because `Text` [7] is much larger than `line` and the loop does not check if [6] operates within the `line`'s buffer bounds, the loop could write out of bounds if a large enough line containing numbers is supplied. This is a straightforward stack-based buffer overflow that could lead to code execution. Moreover, `*text` is assigned later in the same method, so corruption could occur slightly later too.

Allowed characters for `IsNumber` are the following:

```
#define IsNumber( x ) ( ( ( (x) &gt;= &#39;0&#39; ) &amp;&amp; ( (x)
&lt;=&#39;9&#39; ) )   \
                       || ( (x) == &#39;-&#39; ) || ( (x) == &#39;+&#39; )  || ( (x)
== &#39;.&#39; ) )
```

Note that this method is also used while parsing an Excellon file. In that case, this same issue can be triggered via `EXCELLON_IMAGE::LoadFile`, `Execute_EXCELLON_G_Command` or `Execute_Drill_Command`, all eventually calling `ReadIJCoord`.

Crash Information

```
==1399==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fffffff8760 at
pc 0x7fffed7dca70 bp 0x7fffffff85e0 sp 0x7fffffff85d0
WRITE of size 1 at 0x7fffffff8760 thread T0
    #0 0x7fffed7dca6f in GERBER_FILE_IMAGE::ReadIJCoord(char*&amp;amp;)
src/kicad_1/gerbview/rs274_read_XY_and_IJ_coordinates.cpp:225
    #1 0x7fffed7da527 in GERBER_FILE_IMAGE::LoadGerberFile(wxString const&amp;amp;)
src/kicad_1/gerbview/readgerb.cpp:192
    #2 0x7fffed7d8e09 in GERBVIEW_FRAME::Read_GERBER_File(wxString const&amp;amp;)
src/kicad_1/gerbview/readgerb.cpp:58
    #3 0x7fffed783cd4 in GERBVIEW_FRAME::LoadListOfGerberAndDrillFiles(wxString
const&amp;amp;, wxArrayString const&amp;amp;, std::vector&amp;lt;int, std::allocator&amp;lt;int&amp;gt;
&amp;gt; const*) src/kicad_1/ger
bview/files.cpp:293
    #4 0x7fffed780004 in GERBVIEW_FRAME::LoadGerberFiles(wxString const&amp;amp;)
src/kicad_1/gerbview/files.cpp:199
    #5 0x7fffed7acfb5 in GERBVIEW_FRAME::OpenProjectFiles(std::vector&amp;lt;wxString,
std::allocator&amp;lt;wxString&amp;gt; &amp;gt; const&amp;amp;, int)
src/kicad_1/gerbview/gerbview_frame.cpp:273
    #6 0x55555561eae6 in PGM_SINGLE_TOP::OnPgmInit()
src/kicad_1/common/single_top.cpp:428
    #7 0x555555625b0c in APP_SINGLE_TOP::OnInit() (gerbview+0xd1b0c)
    #8 0x5555556245c1 in wxAppConsoleBase::CallOnInit() (gerbview+0xd05c1)
    #9 0x7ffff6a38799 in wxEntry(int&amp;amp;, wchar_t**) (/lib/x86_64-linux-
gnu/libwx_baseu-3.0.so.0+0x113799)
    #10 0x55555561d75a in main src/kicad_1/common/single_top.cpp:269
    #11 0x7ffff509e0b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x270b2)
    #12 0x55555561d2ed in _start (gerbview+0xc92ed)

Address 0x7fffffff8760 is located in stack of thread T0 at offset 288 in frame
    #0 0x7fffed7dc437 in GERBER_FILE_IMAGE::ReadIJCoord(char*&amp;amp;)
src/kicad_1/gerbview/rs274_read_XY_and_IJ_coordinates.cpp:194

  This frame has 1 object(s):
    [32, 288) &amp;#39;line&amp;#39; (line 200) &amp;lt;== Memory access at offset 288 overflows
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
src/kicad_1/gerbview/rs274_read_XY_and_IJ_coordinates.cpp:225 in
GERBER_FILE_IMAGE::ReadIJCoord(char*&amp;amp;)
Shadow bytes around the buggy address:
  0x10007fff7090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff70a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff70b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff70c0: 00 00 00 00 00 00 00 00 f1 f1 f1 f1 00 00 00 00
  0x10007fff70d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=&gt;0x10007fff70e0: 00 00 00 00 00 00 00 00 00 00 00 00[f3]f3 f3 f3
  0x10007fff70f0: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7100: 00 00 00 00 f1 f1 f1 f1 00 00 00 f2 00 00 00 f2
  0x10007fff7110: 00 00 00 f2 f2 f2 00 00 00 00 00 f2 f2 f2 f2 f2
  0x10007fff7120: 00 00 00 00 00 f2 f2 f2 f2 f2 f8 f8 f8 f8 f8 f8
  0x10007fff7130: f2 f2 f2 f2 00 00 00 00 00 00 f2 f2 f2 f2 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
==1399==ABORTING
```

Timeline

2022-02-02 - Vendor Disclosure

2022-02-16 - Public Release

CREDIT

Discovered by Claudio Bozzato of Cisco Talos.
