Talos Vulnerability Report

TALOS-2022-1460

KiCad EDA Gerber Viewer gerber and excellon GCode/Dcode parsing stack-based buffer overflow vulnerability

FEBRUARY 16, 2022

CVE NUMBER

CVE-2022-23947,CVE-2022-23946

Summary

Multiple stack-based buffer overflow vulnerabilities exist in the Gerber Viewer gerber and excellon GCode/Dcode 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. It allows users to design and simulate electronic hardware and offers several tools like a 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, 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];  // [1]

bool GERBER_FILE_IMAGE::LoadGerberFile( const wxString&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( &#34;rt&#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 )  // [2]
            break;

        m_LineNum++;
        text = StrPurge( lineBuffer );

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

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

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

            case &#39;G&#39;:    /* Line type Gxx : command */
                G_command = GCodeNumber( text );                 // [3]
                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 );                // [4]
                Execute_DCODE_Command( text, D_commande );
                break;
```

This method takes a gerber file path, opens it and parses it line-by-line [2], restricting the maximum line length to 1,000,000 bytes [1]. When a line starting with "D" or "G" is encountered, the methods `DCodeNumber` [4] or `GCodeNumber` [3] are called. 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 the `DCodeNumber` method can be reached via an Excellon file in a similar way (via `EXCELLON_IMAGE::LoadFile`), as discussed below.

CVE-2022-23946 - GCodeNumber

```
int GERBER_FILE_IMAGE::GCodeNumber( char*&amp; Text )
{
    int   ii = 0;
    char* text;
    char  line[1024];          // [5]

    if( Text == nullptr )
        return 0;

    Text++;
    text = line;

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

    *text = 0;                 // [7]
    ii    = atoi( line );
    return ii;
}
```

At [5] a `line` buffer of size 1024 bytes is allocated on the stack. The while loop at [6] expects a number inside the line and stores the current character in the `text` buffer (that is, the `line` buffer) [5] and only stops when `Text` does not point to a number anymore.
Because `Text` [1] is much larger than `line` and the loop does not check if it's operating 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 to later in the same method [7], 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; ) )
```

CVE-2022-23947 - DCodeNumber

```
int GERBER_FILE_IMAGE::DCodeNumber( char*&amp; Text )
{
    int   ii = 0;
    char* text;
    char  line[1024];          // [5]

    if( Text == nullptr )
        return 0;

    Text++;
    text = line;

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

    *text = 0;                 // [7]
    ii    = atoi( line );
    return ii;
}
```

At [5] a `line` buffer of size 1024 bytes is allocated on the stack. The while loop at [6] expects a number inside the line and stores the current character in the `text` buffer (that is, the `line` buffer) [5] and only stops when `Text` does not point to a number anymore.
Because `Text` [1] is much larger than `line` and the loop does not check if it's operating 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 to later in the same method [7], 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, when parsing a TCode. In the `EXCELLON_IMAGE` class we find:

```
int TCodeNumber( char*&amp; aText )
{
    return DCodeNumber( aText );
}
```

So in this case, the `DCodeNumber` issue can be triggered via `EXCELLON_IMAGE::Select_Tool`, which eventually calls `DCodeNumber`.

Crash Information

```
=================================================================
==1833==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fffffff8700 at
pc 0x7fffed7df5de bp 0x7fffffff82b0 sp 0x7fffffff82a0
WRITE of size 1 at 0x7fffffff8700 thread T0
    #0 0x7fffed7df5dd in GERBER_FILE_IMAGE::DCodeNumber(char*&amp;)
src/kicad_1/gerbview/rs274d.cpp:435
    #1 0x7fffed7da342 in GERBER_FILE_IMAGE::LoadGerberFile(wxString const&amp;)
src/kicad_1/gerbview/readgerb.cpp:177
    #2 0x7fffed7d8e09 in GERBVIEW_FRAME::Read_GERBER_File(wxString const&amp;)
src/kicad_1/gerbview/readgerb.cpp:58
    #3 0x7fffed783cd4 in GERBVIEW_FRAME::LoadListOfGerberAndDrillFiles(wxString
const&amp;, wxArrayString const&amp;, std::vector&lt;int, std::allocator&lt;int&gt;
&gt; const*) src/kicad_1/gerbview/files.cpp:293
    #4 0x7fffed780004 in GERBVIEW_FRAME::LoadGerberFiles(wxString const&amp;)
src/kicad_1/gerbview/files.cpp:199
    #5 0x7fffed7acfb5 in GERBVIEW_FRAME::OpenProjectFiles(std::vector&lt;wxString,
std::allocator&lt;wxString&gt; &gt; const&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;, 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 0x7fffffff8700 is located in stack of thread T0 at offset 1056 in frame
    #0 0x7fffed7df229 in GERBER_FILE_IMAGE::DCodeNumber(char*&amp;)
src/kicad_1/gerbview/rs274d.cpp:423

  This frame has 1 object(s):
    [32, 1056) &#39;line&#39; (line 426) &lt;== Memory access at offset 1056
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/rs274d.cpp:435
in GERBER_FILE_IMAGE::DCodeNumber(char*&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 00 00 00 00 00 00 00 00
  0x10007fff70d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=&gt;0x10007fff70e0:[f3]f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3 f3
  0x10007fff70f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10007fff7100: f1 f1 f1 f1 00 00 00 f2 00 00 00 f2 00 00 00 f2
  0x10007fff7110: f2 f2 00 00 00 00 00 f2 f2 f2 f2 f2 00 00 00 00
  0x10007fff7120: 00 f2 f2 f2 f2 f2 f8 f8 f8 f8 f8 f8 f2 f2 f2 f2
  0x10007fff7130: 00 00 00 00 00 00 f2 f2 f2 f2 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
  ==1833==ABORTING
```

Timeline

2022-02-14 - Vendor Disclosure

2022-02-16 - Public Release

CREDIT

Discovered by Claudio Bozzato of Cisco Talos.

VULNERABILITY REPORTS

PREVIOUS REPORT

TALOS-2021-1393

NEXT REPORT

TALOS-2022-1453