Talos Vulnerability Report

TALOS-2021-1417

Gerbv pick-and-place rotation parsing use of uninitialized variable vulnerability

JANUARY 31, 2022

CVE NUMBER

CVE-2021-40403

Summary

An information disclosure vulnerability exists in the pick-and-place rotation parsing functionality of Gerbv 2.7.0 and dev (commit b5f1eacd), and Gerbv forked 2.8.0. A specially-crafted pick-and-place file can exploit the missing initialization of a structure to leak memory contents. An attacker can provide a malicious file to trigger this vulnerability.

Tested Versions

Gerbv 2.7.0
Gerbv forked 2.8.0
Gerbv dev (commit b5f1eacd)

Product URLs

Gerbv - https://sourceforge.net/projects/gerbv/ Gerbv forked - https://github.com/gerbv/gerbv

CVSSv3 Score

5.8 - CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

CWE

CWE-456 - Missing Initialization of a Variable

Details

Gerbv is an open-source software that allows users to view RS-274X Gerber files, Excellon drill files and pick-n-place files. These file formats are used in industry to describe the layers of a printed circuit board and are a core part of the manufacturing process.

Some PCB (printed circuit board) manufacturers use software like Gerbv in their web interfaces as a tool to convert Gerber (or other supported) files into images. Users can upload gerber files to the manufacturer website, which are converted to an image to be displayed in the browser, so that users can verify that what has been uploaded matches their expectations. Gerbv can do such conversions using the `-x` switch (export). For this reason, we consider this software as reachable via network without user interaction or privilege requirements.

Gerbv uses the function `gerbv_open_image` to open files. In this advisory we're interested in the pick-and-place file-type.

```
int
gerbv_open_image(gerbv_project_t *gerbvProject, char *filename, int idx, int reload,
                gerbv_HID_Attribute *fattr, int n_fattr, gboolean forceLoadFile)
{
    ...
    dprintf("In open_image, about to try opening filename = %s\n", filename);

    fd = gerb_fopen(filename);
    if (fd == NULL) {
        GERB_COMPILE_ERROR(_("Trying to open \"%s\": %s"),
                        filename, strerror(errno));
        return -1;
    }
    ...
    } else if (pick_and_place_check_file_type(fd, &foundBinary)) {                              // [1]
        dprintf("Found pick-n-place file\n");
        if (!foundBinary || forceLoadFile) {
                if (!reload) {
                        pick_and_place_parse_file_to_images(fd, &parsed_image, &parsed_image2); // [2]
    ...
```

A file is considered of type "pick-and-place" if the function `pick_and_place_check_file_type` [1] returns true. When true, `pick_and_place_parse_file_to_images` is called [2] to parse the input file. Let's first look at the requirements that we need to satisfy to have an input file be recognized as an pick-and-place file:

```
gboolean
pick_and_place_check_file_type(gerb_file_t *fd, gboolean *returnFoundBinary)
{
    ...
    while (fgets(buf, MAXL, fd->fd) != NULL) {
        len = strlen(buf);

        /* First look through the file for indications of its type */

        /* check for non-binary file */
        for (i = 0; i < len; i++) {                                             // [3]
            if (!isprint((int) buf[i]) && (buf[i] != '\r') &&
                (buf[i] != '\n') && (buf[i] != '\t')) {
                found_binary = TRUE;
            }
        }
        ...
        /* Semicolon can be separator too */
        if (g_strstr_len(buf, len, ";")) {
            found_comma = TRUE;
        }

        /* Look for refdes -- This is dumb, but what else can we do? */
        if ((letter = g_strstr_len(buf, len, "R")) != NULL) {
            if (isdigit((int) letter[1])) { /* grab char after R */
                found_R = TRUE;
            }
        }
        if ((letter = g_strstr_len(buf, len, "C")) != NULL) {
            if (isdigit((int) letter[1])) { /* grab char after C */
                found_C = TRUE;
            }
        }
        if ((letter = g_strstr_len(buf, len, "U")) != NULL) {
            if (isdigit((int) letter[1])) { /* grab char after U */
                found_U = TRUE;
            }
        }

        /* Look for board side indicator since this is required
         * by many vendors */
        if (g_strstr_len(buf, len, "top")) {
            found_boardside = TRUE;
        }
        if (g_strstr_len(buf, len, "Top")) {
            found_boardside = TRUE;
        }
        if (g_strstr_len(buf, len, "TOP")) {
            found_boardside = TRUE;
        }
        /* Also look for evidence of "Layer" in header.... */
        if (g_strstr_len(buf, len, "ayer")) {
            found_boardside = TRUE;
        }
        if (g_strstr_len(buf, len, "AYER")) {
            found_boardside = TRUE;
        }

    }
    ...
    if (found_comma && (found_R || found_C || found_U) &&               // [4]
        found_boardside)
        return TRUE;

    return FALSE;

} /* pick_and_place_check_file_type */
```

For an input to be considered a pick-and-place file, the file must first of all contain only printable characters [3]. The other requirements can be gathered by the conditional expression at [4]. An example of a minimal pick-and-place file is the following:

```
# -------------------------------------------
J2,"3 TERM BLOCK","DK ED1602-ND",1501.00,375.00,180,top
```

Though not important for the purposes of the vulnerability itself, note that the checks use `g_strstr_len`, so all those fields can be found anywhere in the file. For example, this file is also recognized as a pick-and-place file, even though it will fail later checks in the execution flow:

```
top,C0
```

After a pick-and-place file has been recognized, `pick_and_place_parse_file_to_images` is called, which in turn calls `pick_and_place_parse_file`. This function parses the pick-and-place file line by line. For each line, a `PnpPartData` structure is built and appended to an array, which is eventually returned. Let's look at the code:

```
GArray *
pick_and_place_parse_file(gerb_file_t *fd)
{
    PnpPartData   pnpPartData;                          // [5]
    int           lineCounter = 0, parsedLines = 0;
    int           ret;
    char          *row[12];                             // [6]
    char          buf[MAXL+2], buf0[MAXL+2];
    char          def_unit[41] = {0,};
    double        tmp_x, tmp_y;
    gerbv_transf_t *tr_rot = gerb_transf_new();
    GArray         *pnpParseDataArray = g_array_new (FALSE, FALSE, sizeof(PnpPartData));
    gboolean foundValidDataRow = FALSE;
    /* Unit declaration for "PcbXY Version 1.0" files as exported by pcb */
    const char *def_unit_prefix = "# X,Y in ";
    ...
```

At [5] we see the `pnpPartData` declaration of type `PnpPartData`:

```
typedef struct {
    char     designator[MAXL];
    char     footprint[MAXL];
    double   mid_x;
    double   mid_y;
    double   ref_x;
    double   ref_y;
    double   pad_x;
    double   pad_y;
    char     layer[MAXL]; /*T is top B is bottom*/
    double   rotation;
    char     comment[MAXL];
    int      shape;
    double   width;
    double   length;
    unsigned int nuf_push;  /* Nuf pushes to estimate stack size */
} PnpPartData;
```

At [6] we see the declaration for the row fields that are going to be populated.
Note that the structure at [5] is not initialized.

```
...
while ( fgets(buf, MAXL, fd->fd) != NULL ) {                                    // [7]
    int len = strlen(buf)-1;
    int i_length = 0, i_width = 0;

    lineCounter += 1; /*next line*/
    if(lineCounter < 2) {                                                       // [8]
        /*
         * TODO in principle column names could be read and interpreted
         * but we skip the first line with names of columns for this time
         */
        continue;
    }
    ...
    if (len <= 11)  {  //lets check a minimum length of 11                     // [9]
        continue;
    }
    ...
    ret = csv_row_parse(buf, MAXL,  buf0, MAXL, row, 11, ',',   CSV_QUOTES);   // [10]

    if (ret > 0) {
        foundValidDataRow = TRUE;
    } else {
        continue;
    }
```

The code loops for each line [7] in the file. At [8] we can see that the first line is skipped, as it's considered a header. At line [9] the code ensures that the line has a minimum size of 12 bytes. There are other sanitization checks, but they are omitted since they're not relevant to this advisory.

At [10] the line is parsed using the `csv_row_parse`. This is a function taken from the `libmba` package, which has been included in gerbv with some modifications. Basically each line is parsed as a CSV, using comma as a separator and allowing a maximum of 11 fields to be populated in the `row` variable. If the CSV parsing is successful, the loop code continues with the following:

```
...
if (row[0] && row[8]) { // here could be some better check for the syntax          // [11]
    snprintf (pnpPartData.designator, sizeof(pnpPartData.designator)-1, "%s", row[0]);
    snprintf (pnpPartData.footprint, sizeof(pnpPartData.footprint)-1, "%s", row[1]);
    snprintf (pnpPartData.layer, sizeof(pnpPartData.layer)-1, "%s", row[8]);
    if (row[10] != NULL) {
        if ( ! g_utf8_validate(row[10], -1, NULL)) {
            gchar * str = g_convert(row[10], strlen(row[10]), "UTF-8", "ISO-8859-1",
                                    NULL, NULL, NULL);
            // I have not decided yet whether it is better to use always
            // "ISO-8859-1" or current locale.
            // str = g_locale_to_utf8(row[10], -1, NULL, NULL, NULL);
            snprintf (pnpPartData.comment, sizeof(pnpPartData.comment)-1, "%s", str);
            g_free(str);
        } else {
            snprintf (pnpPartData.comment, sizeof(pnpPartData.comment)-1, "%s", row[10]);
        }
    }
    ...
    pnpPartData.mid_x = pick_and_place_get_float_unit(row[2], def_unit);
    pnpPartData.mid_y = pick_and_place_get_float_unit(row[3], def_unit);
    pnpPartData.ref_x = pick_and_place_get_float_unit(row[4], def_unit);
    pnpPartData.ref_y = pick_and_place_get_float_unit(row[5], def_unit);
    pnpPartData.pad_x = pick_and_place_get_float_unit(row[6], def_unit);
    pnpPartData.pad_y = pick_and_place_get_float_unit(row[7], def_unit);
    /* This line causes segfault if we accidently starts parsing
     * a gerber file. It is crap crap crap */
    if (row[9])
        sscanf(row[9], "%lf", &pnpPartData.rotation); // no units, always deg      // [13]
}
/* for now, default back to PCB program format
 * TODO: implement better checking for format
 */
else if (row[0] && row[1] && row[2] && row[3] && row[4] && row[5] && row[6]) {     // [12]
    snprintf (pnpPartData.designator, sizeof(pnpPartData.designator)-1, "%s", row[0]);
    snprintf (pnpPartData.footprint, sizeof(pnpPartData.footprint)-1, "%s", row[1]);
    snprintf (pnpPartData.layer, sizeof(pnpPartData.layer)-1, "%s", row[6]);
    pnpPartData.mid_x = pick_and_place_get_float_unit(row[3], def_unit);
    pnpPartData.mid_y = pick_and_place_get_float_unit(row[4], def_unit);
    pnpPartData.pad_x = pnpPartData.mid_x + 0.03;
    pnpPartData.pad_y = pnpPartData.mid_y + 0.03;
    sscanf(row[5], "%lf", &pnpPartData.rotation); // no units, always deg          // [13]
    /* check for coordinate sanity, and abort if it fails
     * Note: this is mainly to catch comment lines that get parsed
     */
    if ((fabs(pnpPartData.mid_x) < 0.001)&&(fabs(pnpPartData.mid_y) < 0.001)) {
        continue;
    }
} else {
    continue;
}
...
```

At [11] and [12] we can notice that two different row formats are accepted: one with 7 fields, and one with at least 9 fields. The difference is minimal, and the issue is the same in both: the `sscanf` at [13] is used to read the `rotation`, and it may fail and not write anything to the destination buffer `pnpPartData.rotation`, which has not been initialized at the beginning of the function [5].

`sscanf` returns the number of items matched and assigned, so in this case it should be equal to `1` when the scan succeeds.

To make `sscanf` fail, it's enough to write an empty string or any non-numeric string in the rotation column. This way, `pnpPartData.rotation` will be left with an uninitialized value from the stack, which will be returned by this function and will later be used to draw the final image. An attacker able to read the resulting rendered image might be able to extract (limited) memory contents by analyzing the object rotation in the rendered image.

Crash Information

```
# valgrind ./gerbv -x png -o out.png pick_and_place_rotation_uninit.min.xy
==863== Memcheck, a memory error detector
==863== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==863== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==863== Command: ./gerbv -x png -o out.png pick_and_place_rotation_uninit.min.xy
==863==
==863== Conditional jump or move depends on uninitialised value(s)
==863==    at 0x5319DD9: sin (s_sin.c:463)
==863==    by 0x158897: gerb_transf_rotate (pick-and-place.c:83)
==863==    by 0x15951A: pick_and_place_parse_file (pick-and-place.c:370)
==863==    by 0x15AB5B: pick_and_place_parse_file_to_images (pick-and-place.c:790)
==863==    by 0x156C92: gerbv_open_image (gerbv.c:538)
==863==    by 0x1561B0: gerbv_open_layer_from_filename_with_color (gerbv.c:249)
==863==    by 0x12D6C9: main (main.c:932)
==863==
==863== Conditional jump or move depends on uninitialised value(s)
==863==    at 0x5319DEF: sin (s_sin.c:465)
==863==    by 0x158897: gerb_transf_rotate (pick-and-place.c:83)
==863==    by 0x15951A: pick_and_place_parse_file (pick-and-place.c:370)
==863==    by 0x15AB5B: pick_and_place_parse_file_to_images (pick-and-place.c:790)
==863==    by 0x156C92: gerbv_open_image (gerbv.c:538)
==863==    by 0x1561B0: gerbv_open_layer_from_filename_with_color (gerbv.c:249)
==863==    by 0x12D6C9: main (main.c:932)
==863==
==863== Conditional jump or move depends on uninitialised value(s)
==863==    at 0x531B8A9: cos (s_sin.c:559)
==863==    by 0x1588AB: gerb_transf_rotate (pick-and-place.c:83)
==863==    by 0x15951A: pick_and_place_parse_file (pick-and-place.c:370)
==863==    by 0x15AB5B: pick_and_place_parse_file_to_images (pick-and-place.c:790)
==863==    by 0x156C92: gerbv_open_image (gerbv.c:538)
==863==    by 0x1561B0: gerbv_open_layer_from_filename_with_color (gerbv.c:249)
==863==    by 0x12D6C9: main (main.c:932)
==863==
==863== Conditional jump or move depends on uninitialised value(s)
==863==    at 0x159587: pick_and_place_parse_file (pick-and-place.c:373)
==863==    by 0x15AB5B: pick_and_place_parse_file_to_images (pick-and-place.c:790)
==863==    by 0x156C92: gerbv_open_image (gerbv.c:538)
==863==    by 0x1561B0: gerbv_open_layer_from_filename_with_color (gerbv.c:249)
==863==    by 0x12D6C9: main (main.c:932)
==863==
==863== Conditional jump or move depends on uninitialised value(s)
==863==    at 0x1595A7: pick_and_place_parse_file (pick-and-place.c:373)
==863==    by 0x15AB5B: pick_and_place_parse_file_to_images (pick-and-place.c:790)
==863==    by 0x156C92: gerbv_open_image (gerbv.c:538)
==863==    by 0x1561B0: gerbv_open_layer_from_filename_with_color (gerbv.c:249)
==863==    by 0x12D6C9: main (main.c:932)
==863==
```

Timeline

2021-11-24 - Vendor Disclosure

2021-12-19 - Vendor Patched

2022-01-31 - Public Release

CREDIT

Discovered by Claudio Bozzato of Cisco Talos.