pnggroup / **libpng** Public

Code Issues 149 Pull requests 28 Discussions Actions Projects

Commit 7ea9eea

ctruta committed 2 weeks ago · ✕ 1 / 4

```
fix: Resolve use-after-free on png_ptr->palette

Give `png_struct` its own independently-allocated copy of the palette
buffer, decoupling it from `info_struct`'s palette. Allocate both
copies with `png_calloc` to zero-fill, because the ARM NEON palette
riffle reads all 256 entries unconditionally.

In function `png_set_PLTE`, `png_ptr->palette` was aliased directly to
`info_ptr->palette`: a single heap buffer shared across two structs
with independent lifetimes. If the buffer was freed through `info_ptr`
(via `png_free_data(PNG_FREE_PLTE)` or a second call to `png_set_PLTE`),
`png_ptr->palette` became a dangling pointer. Subsequent row reads,
performed in `png_do_expand_palette` and in other transform functions,
dereferenced (and in the bit-shift path, wrote to) freed memory.

Also fix `png_set_quantize` to allocate an owned copy of the caller's
palette rather than aliasing the user pointer, so that the unconditional
free in `png_read_destroy` does not free unmanaged memory.
```

libpng18 · v1.6.56

1 parent a3a2144 commit 7ea9eea

5 files changed +35 -29 lines changed

Top

Filter files…

- pngread.c
- pngrtran.c
- pngrutil.c
- pngset.c
- pngwrite.c

5 files changed +35 -29 lines changed

Search within code

pngread.c

```
@@ -779,12 +779,11 @@ png_read_destroy(png_structrp png_ptr)
779  779     png_ptr->quantize_index = NULL;
780  780   #endif
781  781
782      -    if ((png_ptr->free_me & PNG_FREE_PLTE) != 0)
783      -    {
784      -       png_zfree(png_ptr, png_ptr->palette);
785      -       png_ptr->palette = NULL;
786      -    }
787      -    png_ptr->free_me &= ~PNG_FREE_PLTE;
     782  +    /* png_ptr->palette is always independently allocated (not aliased
     783  +     * with info_ptr->palette), so free it unconditionally.
     784  +     */
     785  +    png_free(png_ptr, png_ptr->palette);
     786  +    png_ptr->palette = NULL;
788  787
789  788   #if defined(PNG_tRNS_SUPPORTED) || \
790  789       defined(PNG_READ_EXPAND_SUPPORTED) ||
          defined(PNG_READ_BACKGROUND_SUPPORTED)
```

pngrtran.c

```
@@ -814,7 +814,13 @@ png_set_quantize(png_structrp png_ptr, png_colorp
palette,
814  814     }
815  815     if (png_ptr->palette == NULL)
816  816     {
817      -       png_ptr->palette = palette;
     817  +       /* Allocate an owned copy rather than aliasing the caller's pointer,
     818  +        * so that png_read_destroy can free png_ptr->palette unconditionally.
     819  +        */
     820  +       png_ptr->palette = png_voidcast(png_colorp, png_calloc(png_ptr,
     821  +           PNG_MAX_PALETTE_LENGTH * (sizeof (png_color))));
     822  +       memcpy(png_ptr->palette, palette, (unsigned int)num_palette *
     823  +           (sizeof (png_color)));
818  824     }
819  825     png_ptr->num_palette = (png_uint_16)num_palette;
820  826
```

pngrutil.c

```
@@ -1053,19 +1053,6 @@ png_handle_PLTE(png_structrp png_ptr, png_inforp info_ptr, png_uint_32 length)
1053 1053          /* A valid PLTE chunk has been read */
1054 1054          png_ptr->mode |= PNG_HAVE_PLTE;
1055 1055
1056      -       /* TODO: png_set_PLTE has the side effect of setting png_ptr->palette to
1057      -        * its own copy of the palette.  This has the side effect that when
1058      -        * png_start_row is called (this happens after any call to
1059      -        * png_read_update_info) the info_ptr palette gets changed.  This is
1060      -        * extremely unexpected and confusing.
1061      -        *
1062      -        * REVIEW: there have been consistent bugs in the past about gamma and
1063      -        * similar transforms to colour mapped images being useless because the
1064      -        * modified palette cannot be accessed because of the above.
1065      -        *
1066      -        * CONSIDER: Fix this by not sharing the palette in this way.  But does
1067      -        * this completely fix the problem?
1068      -        */
1069 1056          png_set_PLTE(png_ptr, info_ptr, palette, num);
1070 1057          return handled_ok;
1071 1058       }
```

pngset.c

```
@@ -776,28 +776,38 @@ png_set_PLTE(png_structrp png_ptr, png_inforp info_ptr,
776 776          png_error(png_ptr, "Invalid palette");
777 777       }
778 778
779      -    /* It may not actually be necessary to set png_ptr->palette here;
780      -     * we do it for backward compatibility with the way the png_handle_tRNS
781      -     * function used to do the allocation.
782      -     *
783      -     * 1.6.0: the above statement appears to be incorrect; something has to set
784      -     * the palette inside png_struct on read.
```

```
785      -      */
786  779        png_free_data(png_ptr, info_ptr, PNG_FREE_PLTE, 0);
787  780
788  781        /* Changed in libpng-1.2.1 to allocate PNG_MAX_PALETTE_LENGTH instead
789  782         * of num_palette entries, in case of an invalid PNG file or incorrect
790  783         * call to png_set_PLTE() with too-large sample values.
     784  +      *
     785  +      * Allocate independent buffers for info_ptr and png_ptr so that the
     786  +      * lifetime of png_ptr->palette is decoupled from the lifetime of
     787  +      * info_ptr->palette.  Previously, these two pointers were aliased,
     788  +      * which caused a use-after-free vulnerability if png_free_data freed
     789  +      * info_ptr->palette while png_ptr->palette was still in use by the
     790  +      * row transform functions (e.g. png_do_expand_palette).
     791  +      *
     792  +      * Both buffers are allocated with png_calloc to zero-fill, because
     793  +      * the ARM NEON palette riffle reads all 256 entries unconditionally,
     794  +      * regardless of num_palette.
791  795         */
     796  +     png_free(png_ptr, png_ptr->palette);
792  797        png_ptr->palette = png_voidcast(png_colorp, png_calloc(png_ptr,
793  798            PNG_MAX_PALETTE_LENGTH * (sizeof (png_color))));
     799  +     info_ptr->palette = png_voidcast(png_colorp, png_calloc(png_ptr,
     800  +         PNG_MAX_PALETTE_LENGTH * (sizeof (png_color))));
     801  +     png_ptr->num_palette = info_ptr->num_palette = (png_uint_16)num_palette;
794  802
795  803        if (num_palette > 0)
     804  +     {
     805  +        memcpy(info_ptr->palette, palette, (unsigned int)num_palette *
     806  +            (sizeof (png_color)));
796  807           memcpy(png_ptr->palette, palette, (unsigned int)num_palette *
797  808               (sizeof (png_color)));
     809  +     }
798  810
799      -     info_ptr->palette = png_ptr->palette;
800      -     info_ptr->num_palette = png_ptr->num_palette = (png_uint_16)num_palette;
801  811        info_ptr->free_me |= PNG_FREE_PLTE;
802  812        info_ptr->valid |= PNG_INFO_PLTE;
803  813     }
```

pngwrite.c

```
@@ -1016,6 +1016,10 @@ png_write_destroy(png_structrp png_ptr)
1016 1016    png_ptr->trans_alpha = NULL;
1017 1017  #endif
1018 1018
     1019 +  /* Free the independent copy of the palette owned by png_struct. */
     1020 +  png_free(png_ptr, png_ptr->palette);
     1021 +  png_ptr->palette = NULL;
     1022 +
1019 1023    /* The error handling and memory handling information is left intact at this
1020 1024     * point: the jmp_buf may still have to be freed.  See png_destroy_png_struct
1021 1025     * for how this happens.
```

Comments 0

Please sign in to comment.