author	Szabolcs Nagy <nsz@port70.net>	2026-04-23 21:22:18 +0000
committer	Rich Felker <dalias@aerifal.cx>	2026-10-14 18:48:56 -0400
commit	7813e9119eb5fe681b6750b8a033a0490bca0941 (patch)	
tree	dc8444f8e5dfcc3a592dd62653d17a61f8d773a1	
parent	f607123b7210b52ddfff7c8d5a8a9b84efbe2d6b (diff)	
download	musl-7813e9119eb5fe681b6750b8a033a0490bca0941.tar.gz	

diff options

context: 3

space: include

mode: unified

regex: avoid int overflow in regcomp HEAD master

```
regcomp has many int arith ops that may overflow. before tre_expand_ast
int values are limited by 2x the input length n, e.g.:

  input ERE        int field    max value
  aaaaaaaaaaaa...  position     n
  ()()()()()()...  submatch_id  n/2+1
  ()||||||||||...  num_tags     2n-3

so overflow before expand requires large n (and a lot of memory). later
some values can become big due to repetitions:

  input ERE                     int field that overflows
  (((a{255}){255}){255}){255}   position
  ((a|b*){255,}){255,}          num_transitions

num_transitions and memory use can be big without {k,} repetitions too:

  input ERE        what             max value
  (a|a|a|a...|a)*  num_transitions  n^2/4
  (()|a|a|...|a)*  sum tnfa tags    > n^3/16 - n^2

so memory can run out quickly as n grows (with 31bit va space 1700 long
input is enough even without {k,}) about half is tnfa and half is ast
firstpos/lastpos data. but overflow can happen before allocations fail.

fix early overflow by limiting n. the stack usage is in [n,11n] without
repetitions, usually 7n, so the stack limit got adjusted too. the limit
is such that almost all inputs that worked before still work.

overflows during and after expand are fixed by explicit checks via a
check_add helper and xcalloc instead of xmalloc.
```

Diffstat

-rw-r--r-- src/regex/regcomp.c 41

1 files changed, 35 insertions, 6 deletions

```
diff --git a/src/regex/regcomp.c b/src/regex/regcomp.c
index b4b81968..a7115b12 100644
--- a/src/regex/regcomp.c
+++ b/src/regex/regcomp.c
@@ -1680,6 +1680,12 @@ tre_add_tags(tre_mem_t mem, tre_stack_t *stack, tre_ast_node_t *tree,
    AST to TNFA compilation routines.
 */

+/* check if a+b overflows for a,b in [0,INT_MAX] */
+static int check_add(unsigned a, unsigned b)
+{
```

```
+       return a + b > INT_MAX;
+}
+
 typedef enum {
   COPY_RECURSE,
   COPY_SET_RESULT_PTR
@@ -1731,6 +1737,8 @@ tre_copy_ast(tre_mem_t mem, tre_stack_t *stack, tre_ast_node_t *ast,
                    /* XXX - e.g. [ab] has only one position but two
                       nodes, so we are creating holes in the state space
                       here.  Not fatal, just wastes memory. */
+                   if (check_add(pos, *pos_add))
+                     return REG_ESPACE;
                    pos += *pos_add;
                    num_copied++;
                  }
@@ -1826,6 +1834,8 @@ tre_copy_ast(tre_mem_t mem, tre_stack_t *stack, tre_ast_node_t *ast,
           break;
         }
     }
+  if (check_add(*pos_add, num_copied))
+    return REG_ESPACE;
   *pos_add += num_copied;
   return status;
 }
@@ -1870,6 +1880,8 @@ tre_expand_ast(tre_mem_t mem, tre_stack_t *stack, tre_ast_node_t *ast,
                tre_literal_t *lit= node->obj;
                if (!IS_SPECIAL(lit) || IS_BACKREF(lit))
                  {
+                   if (check_add(lit->position, pos_add))
+                     return REG_ESPACE;
                    lit->position += pos_add;
                    if (lit->position > max_pos)
                      max_pos = lit->position;
@@ -1998,6 +2010,8 @@ tre_expand_ast(tre_mem_t mem, tre_stack_t *stack, tre_ast_node_t *ast,
              }

            iter_depth--;
+           if (check_add(pos_add_total, pos_add - pos_add_last))
+             return REG_ESPACE;
            pos_add_total += pos_add - pos_add_last;
            if (iter_depth == 0)
              pos_add = pos_add_total;
@@ -2010,6 +2024,8 @@ tre_expand_ast(tre_mem_t mem, tre_stack_t *stack, tre_ast_node_t *ast,
         }
     }

+  if (check_add(*position, pos_add_total))
+    return REG_ESPACE;
   *position += pos_add_total;

   /* `max_pos' should never be larger than `*position' if the above
@@ -2711,9 +2727,19 @@ regcomp(regex_t *restrict preg, const char *restrict regex, int cflags)
   /* Parse context. */
   tre_parse_ctx_t parse_ctx;

+  /* Limits: before tre_expand_ast, int values like position, subid or
+   * num_tags are bounded by 2x input length, used stack entries are
+   * bounded by 16x input length. input length is checked upfront to
+   * avoid overflows before expand. later, values that can grow big are
+   * bounds checked via check_add and xcalloc. */
+  const int input_limit = 1024000;
+  const int stack_limit = 8*input_limit;
+  if (strnlen(regex, input_limit) >= input_limit)
+    return REG_ESPACE;
+
   /* Allocate a stack used throughout the compilation process for various
      purposes. */
-  stack = tre_stack_new(512, 1024000, 128);
+  stack = tre_stack_new(512, stack_limit, 128);
   if (!stack)
```

```
      return REG_ESPACE;
    /* Allocate a fast memory allocator. */
@@ -2801,9 +2827,12 @@ regcomp(regex_t *restrict preg, const char *restrict regex, int cflags)
            for example "a*" or "ab*".   Figure out a simple way to detect
            this possibility. */
    tmp_ast_l = tree;
-   tmp_ast_r = tre_ast_new_literal(mem, 0, 0, parse_ctx.position++);
+   tmp_ast_r = tre_ast_new_literal(mem, 0, 0, parse_ctx.position);
    if (tmp_ast_r == NULL)
      ERROR_EXIT(REG_ESPACE);
+   if (check_add(parse_ctx.position, 1))
+     ERROR_EXIT(REG_ESPACE);
+   parse_ctx.position++;

    tree = tre_ast_new_catenation(mem, tmp_ast_l, tmp_ast_r);
    if (tree == NULL)
@@ -2813,22 +2842,22 @@ regcomp(regex_t *restrict preg, const char *restrict regex, int cflags)
    if (errcode != REG_OK)
      ERROR_EXIT(errcode);

-   counts = xmalloc(sizeof(int) * parse_ctx.position);
+   counts = xcalloc(sizeof(int), parse_ctx.position);
    if (counts == NULL)
      ERROR_EXIT(REG_ESPACE);

-   offs = xmalloc(sizeof(int) * parse_ctx.position);
+   offs = xcalloc(sizeof(int), parse_ctx.position);
    if (offs == NULL)
      ERROR_EXIT(REG_ESPACE);

-   for (i = 0; i < parse_ctx.position; i++)
-     counts[i] = 0;
    tre_ast_to_tnfa(tree, NULL, counts, NULL);

    add = 0;
    for (i = 0; i < parse_ctx.position; i++)
      {
        offs[i] = add;
+       if (check_add(add, counts[i] + 1))
+         ERROR_EXIT(REG_ESPACE);
        add += counts[i] + 1;
        counts[i] = 0;
      }
```

generated by cgit v1.2.1 (git 2.18.0) at 2026-11-05 19:07:41 +0000