Alex Smetana

04/ 25/2023

CIS 311-Lab04

Malloc() called with one of the parent function's arguments as a size parameter. I.e…

Expected Output: void function(…, int param, …) { … = malloc(param);}

Code:

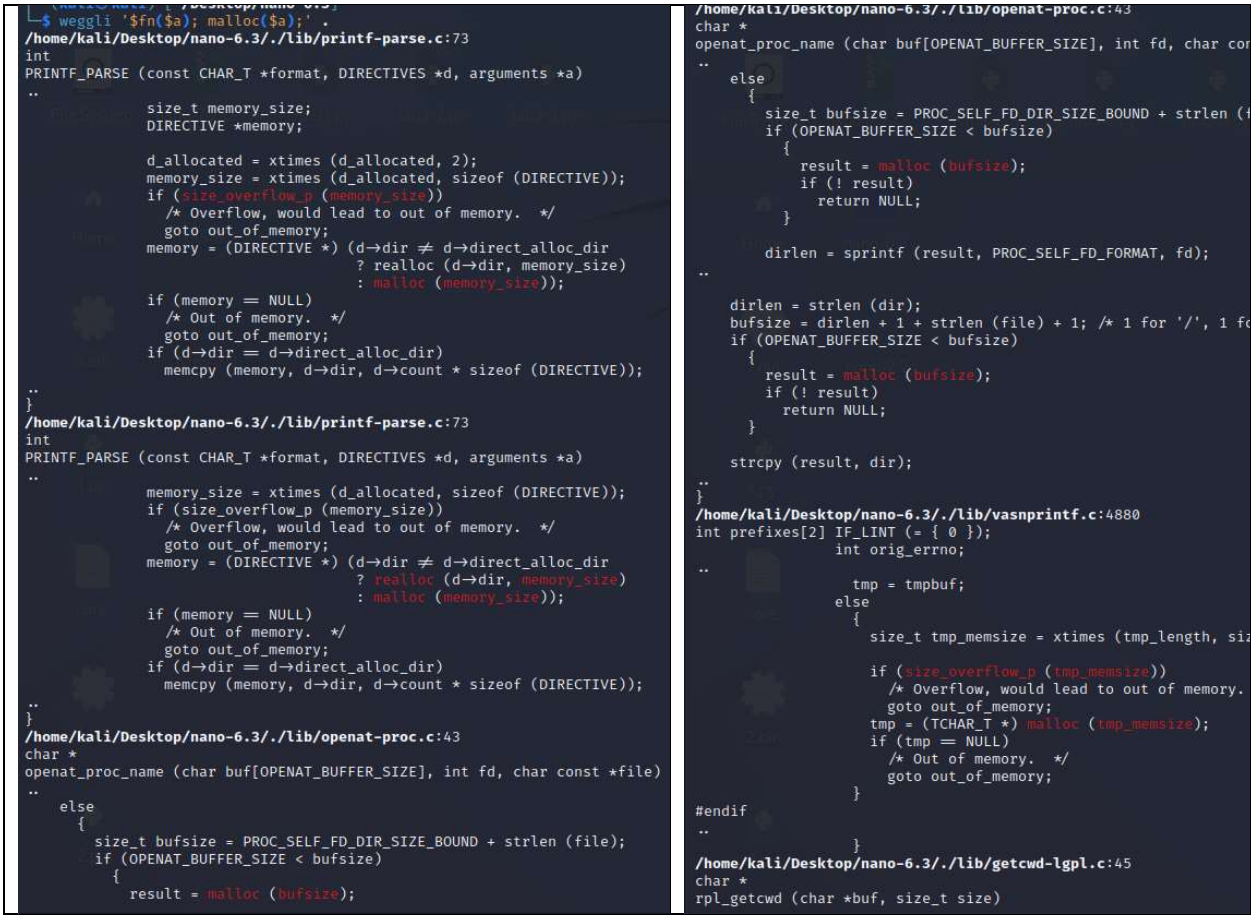
weggli '$fn($a); malloc($a);' .	

Code Walkthrough:

- Weggli '' . – Calls the weggli open source tool.
- $fn($a); - Calls a function with a variable defined as $a.
- Malloc($a); - Calls the malloc function with the same variable $a as defined above.

The goal of the code is to call a malloc function and another function at the same time having the same exact parameters. Firstly, we had to start by calling both of those functions which were called by $fn and malloc(). Then the goal is to set a generic variable that can be called. When the code executes it calls a function and malloc both with the same variable inside of it, this being the variable $a.

Code Executed:



```
└─$ weggli '$fn($a); malloc($a);' .
/home/kali/Desktop/nano-6.3/./lib/printf-parse.c:73
int
PRINTF_PARSE (const CHAR_T *format, DIRECTIVES *d, arguments *a)
{
              size_t memory_size;
              DIRECTIVE *memory;

              d_allocated = xtimes (d_allocated, 2);
              memory_size = xtimes (d_allocated, sizeof (DIRECTIVE));
              if (size_overflow_p (memory_size))
                /* Overflow, would lead to out of memory.  */
                goto out_of_memory;
              memory = (DIRECTIVE *) (d->dir == d->direct_alloc_dir
                                      ? realloc (d->dir, memory_size)
                                      : malloc (memory_size));
              if (memory == NULL)
                /* Out of memory.  */
                goto out_of_memory;
              if (d->dir == d->direct_alloc_dir)
                memcpy (memory, d->dir, d->count * sizeof (DIRECTIVE));
..
}
/home/kali/Desktop/nano-6.3/./lib/printf-parse.c:73
int
PRINTF_PARSE (const CHAR_T *format, DIRECTIVES *d, arguments *a)
{
              memory_size = xtimes (d_allocated, sizeof (DIRECTIVE));
              if (size_overflow_p (memory_size))
                /* Overflow, would lead to out of memory.  */
                goto out_of_memory;
              memory = (DIRECTIVE *) (d->dir == d->direct_alloc_dir
                                      ? realloc (d->dir, memory_size)
                                      : malloc (memory_size));
              if (memory == NULL)
                /* Out of memory.  */
                goto out_of_memory;
              if (d->dir == d->direct_alloc_dir)
                memcpy (memory, d->dir, d->count * sizeof (DIRECTIVE));
..
}
/home/kali/Desktop/nano-6.3/./lib/openat-proc.c:43
char *
openat_proc_name (char buf[OPENAT_BUFFER_SIZE], int fd, char const *file)
{
    else
      {
        size_t bufsize = PROC_SELF_FD_DIR_SIZE_BOUND + strlen (file);
        if (OPENAT_BUFFER_SIZE < bufsize)
          {
            result = malloc (bufsize);
/home/kali/Desktop/nano-6.3/./lib/openat-proc.c:43
char *
openat_proc_name (char buf[OPENAT_BUFFER_SIZE], int fd, char cor
{
    else
      {
        size_t bufsize = PROC_SELF_FD_DIR_SIZE_BOUND + strlen (f
        if (OPENAT_BUFFER_SIZE < bufsize)
          {
            result = malloc (bufsize);
            if (! result)
              return NULL;
          }

        dirlen = sprintf (result, PROC_SELF_FD_FORMAT, fd);
..
    dirlen = strlen (dir);
    bufsize = dirlen + 1 + strlen (file) + 1; /* 1 for '/', 1 fo
    if (OPENAT_BUFFER_SIZE < bufsize)
      {
        result = malloc (bufsize);
        if (! result)
          return NULL;
      }

    strcpy (result, dir);
..
}
/home/kali/Desktop/nano-6.3/./lib/vasnprintf.c:4880
int prefixes[2] IF_LINT (= { 0 });
                int orig_errno;
..
                tmp = tmpbuf;
              else
                {
                  size_t tmp_memsize = xtimes (tmp_length, siz

                  if (size_overflow_p (tmp_memsize))
                    /* Overflow, would lead to out of memory.
                    goto out_of_memory;
                  tmp = (TCHAR_T *) malloc (tmp_memsize);
                  if (tmp == NULL)
                    /* Out of memory.  */
                    goto out_of_memory;
                }
#endif
..
                }
/home/kali/Desktop/nano-6.3/./lib/getcwd-lgpl.c:45
char *
rpl_getcwd (char *buf, size_t size)
```

The zero’th element of one array being copied to the zero’th element of another array. I.e…

Expected Output: array1[0] = array2[0];

Code:

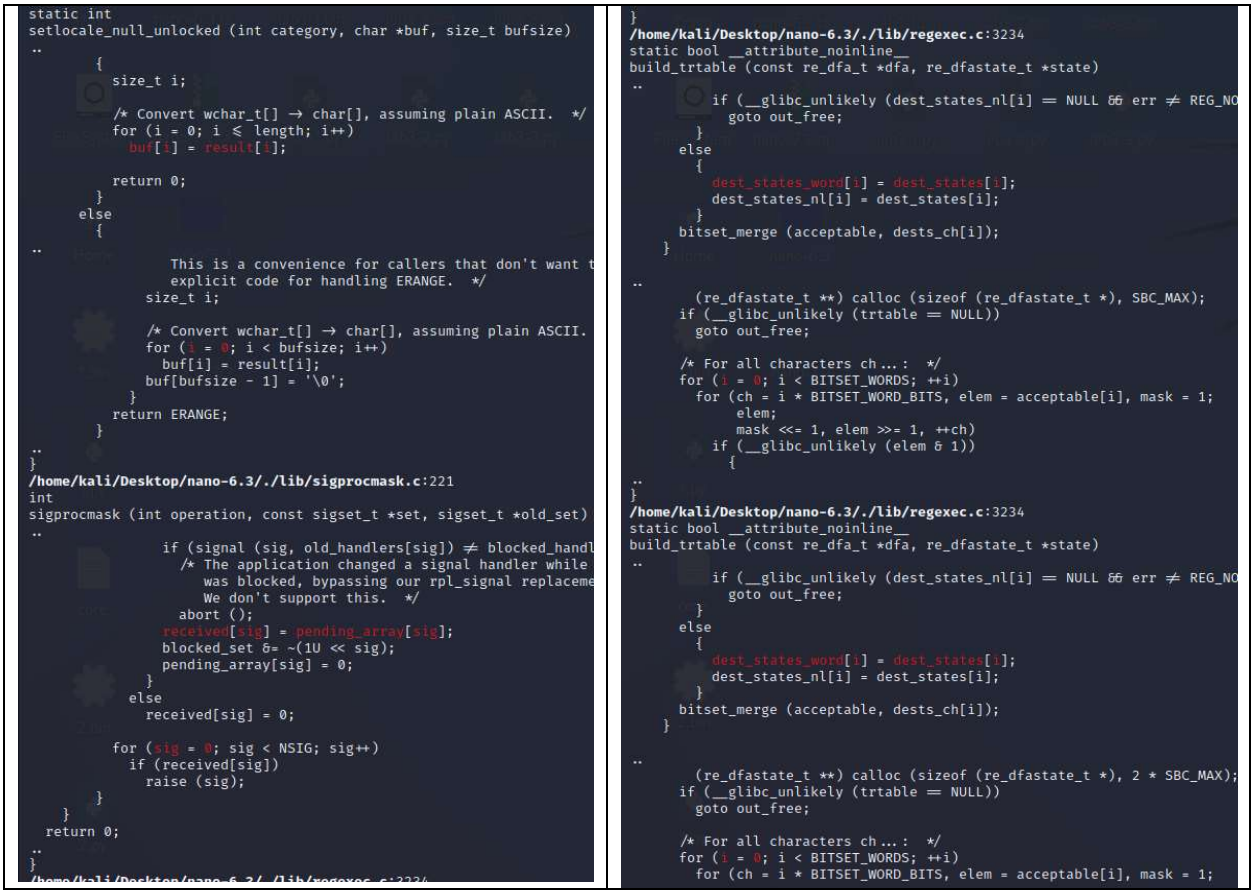
weggli ' $b[$a] = $c[$a]; $a = 0' .	

Code Walkthrough:

- Weggli ‘’ . – Calls the weggli open-source tool.
- $b[$a] = $c[$a]; - Defines 3 variables. $a, $b, and $c and sets them arrays. $b represents the first array and $c represents the second array. $a is the value that is inside of the array that has no value defined yet.
- $a = 0 – We set the value of $a to Zero.

The goal of this code is to call two arrays with one being copied to the 0 element of the others. The first step is to call the arrays. This is done by $b[] and $c[]. Then we need to set the values inside to 0. We can do this by assigning a generic variable $a inside of the array. Then we put this all together and execute the code. When this code is executed, it calls for two different arrays $b and $c, both with the value of $a inside of the array which is zero.

Code Executed:

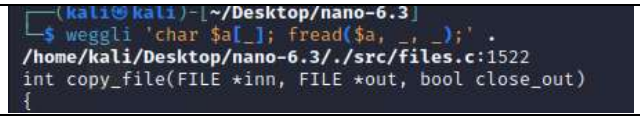


```
static int
setlocale_null_unlocked (int category, char *buf, size_t bufsize)
..
        {
          size_t i;

          /* Convert wchar_t[] → char[], assuming plain ASCII.  */
          for (i = 0; i ≤ length; i++)
            buf[i] = result[i];

          return 0;
        }
      else
        {
..
                This is a convenience for callers that don't want t
                explicit code for handling ERANGE.  */
              size_t i;

              /* Convert wchar_t[] → char[], assuming plain ASCII.
              for (i = 0; i < bufsize; i++)
                buf[i] = result[i];
              buf[bufsize - 1] = '\0';
            }
          return ERANGE;
        }
..
}
/home/kali/Desktop/nano-6.3/./lib/sigprocmask.c:221
int
sigprocmask (int operation, const sigset_t *set, sigset_t *old_set)
..
                if (signal (sig, old_handlers[sig]) ≠ blocked_handl
                  /* The application changed a signal handler while
                     was blocked, bypassing our rpl_signal replaceme
                     We don't support this.  */
                  abort ();
                received[sig] = pending_array[sig];
                blocked_set &= ~(1U << sig);
                pending_array[sig] = 0;
              }
            else
              received[sig] = 0;

          for (sig = 0; sig < NSIG; sig++)
            if (received[sig])
              raise (sig);
        }
    }
  return 0;
..
}
/home/kali/Desktop/nano-6.3/./lib/regexec.c:3234
static bool __attribute_noinline__
build_trtable (const re_dfa_t *dfa, re_dfastate_t *state)
..
          if (__glibc_unlikely (dest_states_nl[i] == NULL && err ≠ REG_NO
            goto out_free;
        }
      else
        {
          dest_states_word[i] = dest_states[i];
          dest_states_nl[i] = dest_states[i];
        }
      bitset_merge (acceptable, dests_ch[i]);
    }

..
        (re_dfastate_t **) calloc (sizeof (re_dfastate_t *), SBC_MAX);
      if (__glibc_unlikely (trtable == NULL))
        goto out_free;

      /* For all characters ch...:  */
      for (i = 0; i < BITSET_WORDS; ++i)
        for (ch = i * BITSET_WORD_BITS, elem = acceptable[i], mask = 1;
             elem;
             mask <<= 1, elem >>= 1, ++ch)
          if (__glibc_unlikely (elem & 1))
            {
..
}
/home/kali/Desktop/nano-6.3/./lib/regexec.c:3234
static bool __attribute_noinline__
build_trtable (const re_dfa_t *dfa, re_dfastate_t *state)
..
          if (__glibc_unlikely (dest_states_nl[i] == NULL && err ≠ REG_NO
            goto out_free;
        }
      else
        {
          dest_states_word[i] = dest_states[i];
          dest_states_nl[i] = dest_states[i];
        }
      bitset_merge (acceptable, dests_ch[i]);
    }

..
        (re_dfastate_t **) calloc (sizeof (re_dfastate_t *), 2 * SBC_MAX);
      if (__glibc_unlikely (trtable == NULL))
        goto out_free;

      /* For all characters ch...:  */
      for (i = 0; i < BITSET_WORDS; ++i)
        for (ch = i * BITSET_WORD_BITS, elem = acceptable[i], mask = 1;
```

fread() called with a local character array variable as the first argument. I.e…

Expected Output: char something[…];fread(something, …, …, …);

Code:

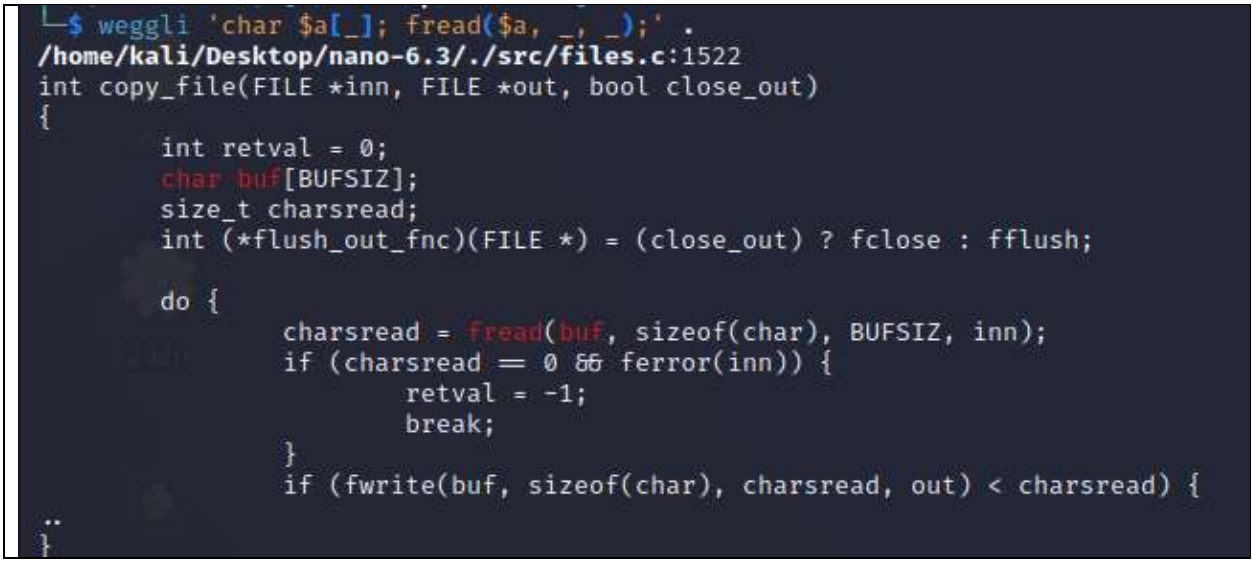
weggli 'char $a[_]; fread($a, _, _);' .	

Code Walkthrough:

- Weggli '' . – Calls the weggli open-source tool.
- Char $a[_] – calls a char $a[_] of a variable with anything in it.
- Fread() – Calls the fread function.
- $a, _, _ - Calls a variable as the first function followed by anything afterwards.

The goal of this code is to call a fread function with a local character array variable as the first argument, followed by anything after. The weggli code indicates that we are using the weggli open-source tool. Fread($a _, _) calls the fread function with $a being required in the first parameter. The "_" at the end of the functions indicates that anything is able to be called afterwards.

Code Executed:



A function returning a pointer which was obtained from malloc().

Expected Output: void * function(…) {void * whatever = malloc(…);return whatever;}

Code:

weggli 'void * $fn(_) {void * $a = malloc(_); return $a;}' .	

Code Walkthrough:

- Weggli ‘’ . – Calls the weggli open-source tool.
- Void $fn(_) – Calls any function with anything inside of it
- Void * $a * malloc(_) – Calls for $ inside of malloc() function.
- return $a – specifies that $a is the return types.

The goal of this code is to call a function that has a return pointer that was obtained from a malloc() function. To start, as always we specify that we are using the weggli open source tool kit using ‘weggli.’ Afterwards, we call ‘void *$fn(_)’ which will call a any type of function. The {void * $a = malloc(_); portion specifies that void * a variable with malloc(_) of any type will be called. Finally, the return$a;} ends the code by specifying that the return type must be the same variable as called earlier. Essentially, this is exactly the same code as the example specifies but replaced with variables.

```
┌──(kali㉿kali)-[~/Desktop/nano-6.3]
└─$ weggli 'void * $fn(_) {void * $a = malloc(_);return $a;}' .
/home/kali/Desktop/nano-6.3/./src/utils.c:286
void *nmalloc(size_t howmuch)
{
        void *section = malloc(howmuch);

        if (section == NULL)
                die(_("Nano is out of memory!\n"));

        return section;
}
/home/kali/Desktop/nano-6.3/./lib/malloc.c:31
void *
rpl_malloc (size_t n)
..
    {
      errno = ENOMEM;
      return NULL;
    }

  void *result = malloc (n);

#if !HAVE_MALLOC_POSIX
  if (result == NULL)
    errno = ENOMEM;
#endif

  return result;
}
```

A function being called with 4 arguments, the last two of which are both ‘0’ (zero). I.e…

Expected Output: whatever(…, …, 0, 0);

Code:

weggli '$fn(_,_,0,0);' .	└─$ weggli '$fn(_,_,0,0);' .

Code Walkthrough:

- Weggli ‘’ . – Calls the weggli open-source tool.
- $fn(_) – Calls any function with anything inside of it
- _, _, 0, 0 – Specifies that the first two inputs can be anything, but the last two have to be zero.

The goal of this query is to call a function where the first two inputs of the query of any value, with the last two inputs being exactly zero. The code firstly starts off by calling the weggli function specifying that we are using the weggli open-source tools. $fn() defines a variable to call any function. Inside the function _,_,0,0 is defined which sets the requirements that the first two inputs can be anything, while making sure that the last two inputs are exactly zero.

```
\/home/kali/Desktop/nano-6.3/./src/nano.c:403
void window_init(void)
{
..
        /* If the terminal is very flat, don't set up a title bar. */
        if (LINES < 3) {
                editwinrows = (ISSET(ZERO) ? LINES : 1);
                /* Set up two subwindows.  If the terminal is just one line,
                 * edit window and status-bar window will cover each other. */
                midwin = newwin(editwinrows, COLS, 0, 0);
                footwin = newwin(1, COLS, LINES - 1, 0);
        } else {
                int toprows = ((ISSET(EMPTY_LINE) && LINES > 6) ? 2 : 1);
                int bottomrows = ((ISSET(NO_HELP) || LINES < 6) ? 1 : 3);

..
}
/home/kali/Desktop/nano-6.3/./src/nano.c:403
void window_init(void)
{
..
#endif
                editwinrows = LINES - toprows - bottomrows + (ISSET(ZERO) ? 1 : 0)

                /* Set up the normal three subwindows. */
                if (toprows > 0)
                        topwin = newwin(toprows, COLS, 0, 0);
                midwin = newwin(editwinrows, COLS, toprows, 0);
                footwin = newwin(bottomrows, COLS, LINES - bottomrows, 0);
        }

        /* In case the terminal shrunk, make sure the status line is clear. */
..
}
```