Bughunting in util-linux

Stefan-Octavian Radu

April 23, 2023

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1 Tools Used

For this analysis I used the memory checker tool that is bundled with `valgrind` and the `cppcheck` static analyser.

1.1 About the Valgrind memory checker

2 Dynamic Analysis

2.1 About the target

My analysis started from the `cal` terminal utility. Running the `valgrind` memory checker on the already installed instanced I could see that a few thousand kilobytes of memory were either lost or not properly freed upon the program exist. `cal` is a simply utility program that runs in the terminal and displays a calender with the current day highlighted using the `ncurses` library. It is part of a bigger suite of CLI utilities and programs bundled together in the `util-linux` repository [**?**]. The repository hosted on Github, which this paper is based on is used for specifically for development.

2.2 Investigation process

The first step was to clone the repository and compile the targeted program. Running `valgrind` on the compiled program leads to the results shown in Listing 1.

```
LEAK SUMMARY:
   definitely lost: 0 bytes in 0 blocks
   indirectly lost: 0 bytes in 0 blocks
     possibly lost: 0 bytes in 0 blocks
   still reachable: 17,760 bytes in 20 blocks
        suppressed: 0 bytes in 0 blocks
```

Listing 1: Innitial Leak Summary

Seeing `still reachable` memory is sign of improper handling and freeing of allocated memory before a program exits. Using the additional flags `--leak-check=full` and `--show-leak-kinds=all` I can get a stack trace of the memory allocation in the first place.

Following the stack trace and testing the code along the way I understood that while initializing the necessary elements needed for colour support, there is also a check performed that verifies if the current terminal in used supports colours.

```
// cal.c
if (colors_init(ctl.colormode, "cal") == 0) {
    ...
// colors.c
if (cc->mode == UL_COLORMODE_UNDEF
    && (ready = colors_terminal_is_ready())) {
    ...
// colors.c
if (setupterm(NULL, STDOUT_FILENO, &ret) == 0
        && ret == 1)
```

Listing 2: Stack Trace

The `setupterm` function is part of the `ncurses` system library. While consulting its manual [**?**] I learned that, as the name suggests, `setupterm` is a routine that handles initialization of various low-level terminal-dependant structures and variables. Upon initialization, the `cur_term` global is set to point to the newly initialized memory segment. Releasing this memory however, is the responsibility of the developer who should call the `del_curterm`. A careful inspection of the code shows that `del_curterm` is not called while using the `cal` program.

2.3 Solution

The solution I came up with was to insure that when `setupterm` is called, `del_curterm` will also be called. For this I used `atexit` from the standard C library which calls a provided function when the program exits. I thus created a wrapper around the `del_curterm` function and pass it as an argument to the `atexit` call as seen in Listing 3.

```
/* atexit() wrapper */
static void colors_del_curterm(void)
{
  del_curterm(cur_term);
}
...

if (setupterm(NULL, STDOUT_FILENO, &ret) == 0
        && ret == 1) {
    ...
    atexit(colors_del_curterm);
}
```

Listing 3: Wrapper for atexit

2.4 Results

Following the proposed change, I recorded its effects by running `valgrind` again on the newly compiled binary. As shown in Listing 4, I managed to reduce the improperly released memory by 9640 bytes in a total of 15 blocks.

```
LEAK SUMMARY:
   definitely lost: 0 bytes in 0 blocks
   indirectly lost: 0 bytes in 0 blocks
     possibly lost: 0 bytes in 0 blocks
   still reachable: 8,120 bytes in 5 blocks
        suppressed: 0 bytes in 0 blocks
```

Listing 4: Leak Summary after changes

Since the problem originated from the `colors.h` library, this change affects not only the `cal` utility, but any other program from the repository which imports `colors.h`. This includes common utilities such as `fdisk`, or `hexdump`.

2.5 Further efforts

As it's obvious from Listing 4, there is still unreleased memory when the process finishes execution. By following the cues from `valgrind` I concluded the problem originate in the `tigetnum("colors");` as well as in low level code in the `ncurses` library. After unsuccessful attempts of fixing the problem from the library level and some more research on the topic I came across an interesting finding in the ncurses FAQ page [**?**]. It seems that reports regarding memory still in use in programs which depend on the `ncurses` library are normal and expected. There are certain chunks of memory which are never freed for performance reasons.

The final verdict regarding the 5 blocks of memory still in use is thus inconclusive. There is a high probability that not properly releasing the respective blocks is intentional, but I couldn't find conclusive evidence for this.

3 Static Analysis