Fixing Old Bugs

There has been a bug that has bothered me for the last 10 years. I first encountered this bug on 10.4 and it involves the man page view "info". If you want to follow along at home this is a very simple crash to reproduce. Open a new terminal window, type in 'info whois' (or the name of any other program that has a man page), then resize the terminal window to be very small. If you are to then expand the window back out to a normal size you will see that info has crashed.

Reproducing these steps with lldb attached produces this:

```
* thread #1: tid = 0x96b4bb, 0x000000010af487c7 info`___lldb_unnamed_fu
frame #0: 0x000000010af487c7 info`___lldb_unnamed_function7$$info + 129
info`___lldb_unnamed_function7$$info + 1291:
 -> 0x10af487c7:  movq   (%r15), %rdi
    0x10af487ca:  movl   $0x1b, %esi
    0x10af487cf:  callq  0x10af62a36               ; symbol stub for: s
    0x10af487d4:  testq  %rax, %rax
```

This snippet of assembly can be translated into the following C code:

```
strchr(%r15, '\033')
```

This is trying to scan a cstring for any occurrences of the escape character '\033' (hex representation: 0x1b).

On Mavericks (10.9), the shipped version of info is 4.8, which was published in 2004. I downloaded the old source code to this to see if I could find the exact crash a bit easier.

After building it from source I repeated the steps to cause the crash and this time lldb greeted me with the following:

```
* thread #1: tid = 0x96d28e, 0x000000010c4f4a7d ginfo`display_update_on
frame #0: 0x000000010c4f4a7d ginfo`display_update_one_window(win=0x0000
    300                   happen if the window is shrunk very small.)  *
    301               if ((entry && entry->inverse)
    302               /* Need to erase the line if it has escape sequen
 -> 303               || (raw_escapes_p && strchr (entry->text, '\033')
    304                 {
    305                   terminal_goto_xy (0, line_index + win->first_
    306                   terminal_clear_to_eol ();
```

This matches the assembly given by the first crash using the version shipped by Apple. The issue is glaring, this conditional statement has two parts that are OR'd but are dependent on each other to be successful.

1. Checking if the local variable `entry` is not NULL, and then if the entry is inverted.
2. Checking if there are escapes and if the entry text contains any escape characters.

Since they are OR'd, even if the first part of the conditional fails on "entry" not being NULL, it will still try to check the text contents of the entry and result in a NULL dereference. To verify this, I looked at `entry` in lldb and sure enough:

```
(lldb) p entry
(DISPLAY_LINE *) $124 = 0x0000000000000000
```

There is the null pointer which will be dereferenced and cause our crash. Now this seems like a straight forward fix, by separating out the conditional logic to evaluate the `entry` value before any other part we avoid the dereference and crash entirely. However, since 4.8 is from 2004, it might be better to check the latest shipped version for this crash instead and then file a bug report against apple to update the binary to stop this crash.

The latest official version is 5.2, which was published in September 2013. I downloaded the source code to that to see if this was fixed. After repeating the steps I found something new:

```
* thread #1: tid = 0x970058, 0x000000010a717db5 ginfo`window_scan_line(
frame #0: 0x000000010a717db5 ginfo`window_scan_line(win=0x00007fb6f9503
    1785              void *closure)
    1786    {
    1787      mbi_iterator_t iter;
 -> 1788      long cpos = win->line_starts[line] - win->node->contents;
    1789      int delim = 0;
    1790      char *endp;
    1791
```

Same crash reason, but caused by different code this time. It is possible that since it had gone through a major version update that it could have entirely new code. Another bad memory access, so first thing to check is the variable `win`.

```
(lldb) p win
(WINDOW *) $125 = 0x00007fb6f9503db0
```

That looks ok, but to make sure we are looking at a valid pointer, we should print the dereferenced value in the debugger to be sure, then check what we are accessing on that line.

```
(lldb) p *win
(WINDOW) $126 = {
    next = 0x0000000000000000
    prev = 0x0000000000000000
    width = 97
    height = 0
    first_row = 0
    goal_column = 18446744073709551615
    keymap = 0x00007fb6fa01d200
    node = 0x00007fb6f9700370
    pagetop = 27
    point = 1000
    line_map = {
        node = 0x00007fb6f9700370
        nline = 0
        size = 80
        used = 0
        map = 0x00007fb6f96025c0
    }
    modeline = 0x00007fb6f9700990 "-----Info: (*manpages*)whois, 144 li
    line_starts = 0x0000000000000000
    line_count = 0
    log_line_no = 0x00007fb6fb003600
    flags = 5
}
```

Again, another null-dereference, this time from the member `line_starts`. Now we are in the most recent version of code fixes can be added to stop these crashes from happening. By adding a new conditional check to the start of this function:

####Bug #1

Original:

```
int
window_scan_line (WINDOW *win, int line, int phys,
      void (*fun) (void *closure, long cpos, size_t replen),
      void *closure)
{
    mbi_iterator_t iter;
    long cpos = win->line_starts[line] - win->node->contents;
    int delim = 0;
    char *endp;
    /*
    ...
    */
    return cpos;
}
```

Patch:

```
int
window_scan_line (WINDOW *win, int line, int phys,
      void (*fun) (void *closure, long cpos, size_t replen),
      void *closure)
{
    mbi_iterator_t iter;
    long cpos = 0;
    if (win->line_starts != NULL) { // the new check to stop null-deref
        cpos = win->line_starts[line] - win->node->contents;
        /*
        ...
        */
    }
    return cpos;
}
```

This has fixed the first bug. To verify that it has been successfully patched, we are going to run through the steps again.

```
* thread #1: tid = 0x9723b1, 0x00007fff8aeec866 libsystem_kernel.dylib`
frame #0: 0x00007fff8aeec866 libsystem_kernel.dylib`__pthread_kill + 10
libsystem_kernel.dylib`__pthread_kill + 10:
 -> 0x7fff8aeec866:  jae    0x7fff8aeec870            ; __pthread_kill
    0x7fff8aeec868:  movq   %rax, %rdi
    0x7fff8aeec86b:  jmp    0x7fff8aee9175            ; cerror_nocancel
    0x7fff8aeec870:  retq
```

Well, this time it is crashing outside of the scope of the process. Running a backtrace on the crash should reveal more information:

```
(lldb) bt
* thread #1: tid = 0x9723b1, 0x00007fff8aeec866 libsystem_kernel.dylib`
  * frame #0: 0x00007fff8aeec866 libsystem_kernel.dylib`__pthread_kill
    frame #1: 0x00007fff8e70035c libsystem_pthread.dylib`pthread_kill +
    frame #2: 0x00007fff8ad44b1a libsystem_c.dylib`abort + 125
    frame #3: 0x00007fff92fa607f libsystem_malloc.dylib`free + 411
    frame #4: 0x000000010bf2d043 ginfo`window_new_screen_size(width=0,
    frame #5: 0x000000010bf2aba3 ginfo`reset_info_window_sizes + 83 at
    frame #6: 0x000000010bf2aa42 ginfo`info_signal_proc(sig=<unavailabl
    frame #7: 0x00007fff8e5ca5aa libsystem_platform.dylib`_sigtramp + 2
    frame #8: 0x00007fff8aeed9f1 libsystem_kernel.dylib`read + 9
    frame #9: 0x00007fff784e9420 libsystem_c.dylib`__strerror_ebuf + 16
    frame #10: 0x000000010bf229ac ginfo`info_read_and_dispatch + 204 at
    frame #11: 0x000000010bf22887 ginfo`display_startup_message_and_sta
    frame #12: 0x000000010bf22868 ginfo`display_startup_message_and_sta
    frame #13: 0x000000010bf1a56e ginfo`single_file(argc=<unavailable>,
    frame #14: 0x000000010bf19ef5 ginfo`main(argc=<unavailable>, argv=0
    frame #15: 0x00007fff8e8c15fd libdyld.dylib`start + 1
```

The backtrace shows that the problem is being caused in frame 4, and the resulting frames hint at an over-release problem. Jumping back to frame 4 in the debugger seals it.

```
(lldb) frame select 4
frame #4: 0x000000010bf2d043 ginfo`window_new_screen_size(width=0, heig
    139               windows->height = 0;
    140               free (windows->line_starts);
 -> 141               free (windows->log_line_no);
    142               windows->line_starts = NULL;
    143               windows->line_count = 0;
    144               break;
```

By checking the "windows" variable in that scope:

```
(lldb) p *windows
(WINDOW) $128 = {
    next = 0x0000000000000000
    prev = 0x0000000000000000
    width = 105
    height = 0
    first_row = 0
    goal_column = 18446744073709551615
    keymap = 0x00007f8355006a00
    node = 0x00007f8353d00370
    pagetop = 22
    point = 33214047251857408
    line_map = {
        node = 0x00007f8353d00370
        nline = 0
        size = 80
        used = 0
        map = 0x00007f8353d058d0
    }
    modeline = 0x00007f8353e00410 "-----Info: (*manpages*)whois, 144 li
    line_starts = 0x0000000000000000
    line_count = 0
    log_line_no = 0x00007f8355800600
    flags = 5
}
```

When `windows->line_starts` is NULL, then the value assigned to `windows->log_line_no` seems to not be allocated, just assigned a reference. Attempting to free a non-allocated pointer is a bad idea, so it looks like there is some more NULL checks that must be added.

####Bug #2

Original:

```
windows->height = 0;
free (windows->line_starts);
free (windows->log_line_no);
windows->line_starts = NULL;
windows->line_count = 0;
break;
```

Patch:

```
windows->height = 0;
if (windows->line_starts) {
    free (windows->line_starts);
    windows->line_starts = NULL;
    free (windows->log_line_no);
}
windows->log_line_no = NULL;
windows->line_count = 0;
break;
```

This code fixes two bugs that are basically the same root cause. The crash is caused when `windows->line_starts` is NULL, the `windows->log_line_no` isn't allocated memory (just a reference to zero) and will cause a crash when freed.

Running it again with these patches causes yet another set of crashes:

Crash #1:

```
* thread #1: tid = 0x98265e, 0x0000000104338d91 ginfo`display_node_text
frame #0: 0x0000000104338d91 ginfo`display_node_text(closure=<unavailab
    135          the line from the screen first.  Why, I don't know.
    136          (But don't do this if we have no visible entries, as c
    137          happen if the window is shrunk very small.)  */
 -> 138           if (entry->inverse
    139           /* Need to erase the line if it has escape sequences.
    140           || (raw_escapes_p && mbschr (entry->text, '\033') !=
    141         {
```

Crash #2:

```
* thread #1: tid = 0x982d2a, 0x0000000106974dd1 ginfo`display_node_text
frame #0: 0x0000000106974dd1 ginfo`display_node_text(closure=<unavailab
    142           terminal_goto_xy (0, win->first_row + pline_index);
    143           terminal_clear_to_eol ();
    144           entry->inverse = 0;
 -> 145           entry->text[0] = '\0';
    146           entry->textlen = 0;
    147         }
```

This is the result of a fall-through case in `entry->inverse`. The struct `DISPLAY_LINE` is defined as:

```
typedef struct {
    char *text;         /* Text of the line as it appears. */
    int textlen;            /* Printable Length of TEXT. */
    int inverse;            /* Non-zero means this line is inverse. */
} DISPLAY_LINE;
```

However the member `inverse` is only ever set to `0` or `1`, never any other number. The check on line 138 in the first crash is a bad access of `entry->inverse`, and the second crash is a result of the first conditional expression of that if statement passing due to the value of `entry->inverse` being something other than zero. An explicit check against the value of `inverse` will mitigate these crashes.

####Bug #3

Original:

```
/* If the screen line is inversed, then we have to clear
 the line from the screen first.  Why, I don't know.
 (But don't do this if we have no visible entries, as can
 happen if the window is shrunk very small.)  */
  if (entry->inverse
  /* Need to erase the line if it has escape sequences.  */
  || (raw_escapes_p && mbschr (entry->text, '\033') != 0))
{
```

Patch:

```
/* If the screen line is inversed, then we have to clear
 the line from the screen first.  Why, I don't know.
 (But don't do this if we have no visible entries, as can
 happen if the window is shrunk very small.)  */
  if (entry->inverse == 1
  /* Need to erase the line if it has escape sequences.  */
  || (raw_escapes_p && mbschr (entry->text, '\033') != 0))
{

```

With the fourth bug patched it is directly onto the next crasher:

```
* thread #1: tid = 0x98caa1, 0x00007fff8e5caa46 libsystem_platform.dyli
frame #0: 0x00007fff8e5caa46 libsystem_platform.dylib`_platform_strchr
libsystem_platform.dylib`_platform_strchr + 38:
 -> 0x7fff8e5caa46:  movdqa (%rdi), %xmm2
    0x7fff8e5caa4a:  pcmpeqb %xmm2, %xmm1
    0x7fff8e5caa4e:  pcmpeqb %xmm0, %xmm2
    0x7fff8e5caa52:  por    %xmm1, %xmm2
(lldb) bt
* thread #1: tid = 0x98caa1, 0x00007fff8e5caa46 libsystem_platform.dyli
    * frame #0: 0x00007fff8e5caa46 libsystem_platform.dylib`_platform_s
      frame #1: 0x000000010a6818d8 ginfo`mbschr(string=0x20612d20202020
      frame #2: 0x000000010a662db2 ginfo`display_node_text(closure=<una
      frame #3: 0x000000010a67f75b ginfo`process_node_text(win=0x00007f
      frame #4: 0x000000010a662c44 ginfo`display_update_one_window(win=
      frame #5: 0x000000010a662b75 ginfo`display_update_display(window=
      frame #6: 0x000000010a67bba3 ginfo`reset_info_window_sizes [inlin
      frame #7: 0x000000010a67bb73 ginfo`reset_info_window_sizes + 83 a
      frame #8: 0x000000010a67ba12 ginfo`info_signal_proc(sig=<unavaila
      frame #9: 0x00007fff8e5ca5aa libsystem_platform.dylib`_sigtramp +
      frame #10: 0x00007fff8aeed9f1 libsystem_kernel.dylib`read + 9
      frame #11: 0x00007fff784e9420 libsystem_c.dylib`__strerror_ebuf +
      frame #12: 0x000000010a67397c ginfo`info_read_and_dispatch + 204
      frame #13: 0x000000010a673857 ginfo`display_startup_message_and_s
      frame #14: 0x000000010a673838 ginfo`display_startup_message_and_s
      frame #15: 0x000000010a66b53e ginfo`single_file(argc=<unavailable
      frame #16: 0x000000010a66aec5 ginfo`main(argc=<unavailable>, argv
      frame #17: 0x00007fff8e8c15fd libdyld.dylib`start + 1
```

Now back to the original bug with using `strchr` to check for escape characters in the text. This time instead of a NULL dereference the cstring pointer being handed to it is an invalid memory address. Walking back through the frames reveals some more info:

```
(lldb) frame select 1
frame #1: 0x000000010a6818d8 ginfo`mbschr(string=0x20612d2020202020, c=
    48            return NULL;
    49          }
    50        else
 -> 51          return strchr (string, c);
    52      }
(lldb) frame select 2
frame #2: 0x000000010a662db2 ginfo`display_node_text(closure=<unavailab
    137          happen if the window is shrunk very small.)  */
    138           if (entry->inverse == 1
    139           /* Need to erase the line if it has escape sequences.
 -> 140           || (raw_escapes_p && mbschr (entry->text, '\033') !=
    141         {
    142           terminal_goto_xy (0, win->first_row + pline_index);
    143           terminal_clear_to_eol ();
(lldb) frame select 3
frame #3: 0x000000010a67f75b ginfo`process_node_text(win=0x00007f8eda40
    1645
 -> 1646          rc = fun (closure, line_index, logline_index,
    1647                mbi_cur_ptr (iter) - in_index,
    1648                printed_line, pl_index, pl_count);
    1649
```

Inside of frame 2, the variable `entry` is defined as:

```
struct display_node_closure *dn = closure;
WINDOW *win = dn->win;
DISPLAY_LINE **display = dn->display;
DISPLAY_LINE *entry = display[win->first_row + pline_index];
```

In the case of this crash, `win->first_row == 0` and `pline_index == 8`. This would be the contents of `closure` in frame 2, corresponding to the struct:

```
(struct display_node_closure *)closure =>

struct display_node_closure {
    WINDOW *win;
    DISPLAY_LINE **display;
};
```

The contents of the member `display` is an array of pointers to instances of `DISPLAY_LINE`:

```
typedef struct {
    char *text;         /* Text of the line as it appears. */
    int textlen;            /* Printable Length of TEXT. */
    int inverse;            /* Non-zero means this line is inverse. */
} DISPLAY_LINE;
```

When looking at these contents in memory:

```
[index] (DISPLAY_LINE **)
[0]     40 00 E0 4A AD 7F 00 00 => 0x00007FAD4AE00040
[1]     C0 00 E0 4A AD 7F 00 00 => 0x00007FAD4AE000C0
[2]     40 01 E0 4A AD 7F 00 00 => 0x00007FAD4AE00140
[3]     C0 01 E0 4A AD 7F 00 00 => 0x00007FAD4AE001C0
[4]     40 02 E0 4A AD 7F 00 00 => 0x00007FAD4AE00240
[5]     C0 02 E0 4A AD 7F 00 00 => 0x00007FAD4AE002C0
[6]     40 03 E0 4A AD 7F 00 00 => 0x00007FAD4AE00340
[7]     00 00 00 00 00 00 00 00 => 0x0, signifying the end of the array

Index 0: (Offset 0x00007FAD4AE00040)
50 00 E0 4A AD 7F 00 00 => 0x00007FAD4AE00050 (offset of the text membe
4D 00 00 00 => textlen == 77
00 00 00 00 => inverse == 0
20 20 20 20 20 20 20 20 20 20 20 20 4E 4F 54 45 21 20 20 54 68 65 20
    => text == "             NOTE!  The registration of these domains i

Index 1: (Offset 0x00007FAD4AE000C0)
D0 00 E0 4A AD 7F 00 00 => 0x00007FAD4AE000D0 (offset of the text membe
4D 00 00 00 => textlen == 77
00 00 00 00 => inverse == 0
20 20 20 20 20 20 20 20 20 20 20 20 20 6F 66 20 69 6E 64 65 70 65 6E 64
    => text == "             of independent and competing registrars.

Index 2: (Offset 0x00007FAD4AE00140)
50 01 E0 4A AD 7F 00 00 => 0x00007FAD4AE00150 (offset of the text membe
4A 00 00 00 => textlen == 74
00 00 00 00 => inverse == 0
20 20 20 20 20 20 20 20 20 20 20 20 69 6E 66 6F 72 6D 61 74 69 6F 6E
    => text == "             information on domains registered by organ

Index 3: (Offset 0x00007FAD4AE001C0)
D0 01 E0 4A AD 7F 00 00 => 0x00007FAD4AE001D0 (offset of the text membe
4B 00 00 00 => textlen == 75
00 00 00 00 => inverse == 0
20 20 20 20 20 20 20 20 20 20 20 20 4E 65 74 77 6F 72 6B 20 53 6F 6C
    => text == "             Network Solutions, Inc.  Also, note that t

Index 4: (Offset 0x00007FAD4AE00240)
50 02 E0 4A AD 7F 00 00 => 0x00007FAD4AE00250 (offset of the text membe
4C 00 00 00 => textlen == 76
00 00 00 00 => inverse == 0
20 20 20 20 20 20 20 20 20 20 20 20 28 77 68 6F 69 73 2E 69 6E 74 65
    => text == "             (whois.internic.net) is no longer handled

Index 5: (Offset 0x00007FAD4AE002C0)
D0 02 E0 4A AD 7F 00 00 => 0x00007FAD4AE002D0 (offset of the text membe
3D 00 00 00 => textlen == 61
00 00 00 00 => inverse == 0
20 20 20 20 20 20 20 20 20 20 20 20 49 6E 63 2E 20 20 46 6F 72 20 64
```

```
    => text == "            Inc.  For details, see http://www.internic

Index 6: (Offset 0x00007FAD4AE00340)
50 03 E0 4A AD 7F 00 00 => 0x00007FAD4AE00350 (offset of the text membe
00 00 00 00 => textlen == 0
00 00 00 00 => inverse == 0
00
    => text == "" (empty string)
```

Referencing how we access the current entry:

```
DISPLAY_LINE *entry = display[win->first_row + pline_index]; // => disp
```

This puts the `DISPLAY_LINE*` entry beyond the bounds of the array and instead, accessing the pointer to the first string entry contents `0x00007FAD4AE00050`. This results in a valid pointer but of the wrong type, resulting in a bad memory access when it performs `strchr` against the first 8 bytes the string rather than accessing the 8 bytes that would be the pointer to the string contents.

####Bug #4

Original:

```
struct display_node_closure *dn = closure;
WINDOW *win = dn->win;
DISPLAY_LINE **display = dn->display;
DISPLAY_LINE *entry = display[win->first_row + pline_index];
```

Patch:

```
struct display_node_closure *dn = closure;
WINDOW *win = dn->win;
DISPLAY_LINE **display = dn->display;
int index_count = 0;
while (display[index_count] != NULL) {
    index_count++;
}
int entry_index = win->first_row + pline_index;
if (entry_index > index_count) {
    return 0;
}

DISPLAY_LINE *entry = display[entry_index];
```

This patch is a serious hack, but lacking any way to reliably check the number of indexed entries in the array it is the only thing I can come up with to ensure the valid indexing.

Repeating the steps turns up another crash:

```
* thread #1: tid = 0x99ec07, 0x0000000108031c62 ginfo`display_update_on
frame #0: 0x0000000108031c62 ginfo`display_update_one_window(win=0x0000
    261             DISPLAY_LINE *entry = display[win->first_row + line
    262
    263             /* If this line has text on it then make it go away
 -> 264             if (entry && entry->textlen)
    265             {
    266                 entry->textlen = 0;
    267                 entry->text[0] = '\0';
```

This is another invalid indexing bug.

####Bug #5

Original:

```
for (; line_index < win->height; line_index++)
{
    DISPLAY_LINE *entry = display[win->first_row + line_index];

    /* If this line has text on it then make it go away. */
    if (entry && entry->textlen)
    {
        entry->textlen = 0;
        entry->text[0] = '\0';

        terminal_goto_xy (0, win->first_row + line_index);
        terminal_clear_to_eol ();
    }
}
```

Patch:

```
for (; line_index < win->height; line_index++)
{
    int index_count = 0;
    while (display[index_count] != NULL) {
        index_count++;
    }
    int entry_index = win->first_row + line_index;
    if (entry_index > index_count) {
        break;
    }
    DISPLAY_LINE *entry = display[entry_index];

    /* If this line has text on it then make it go away. */
    if (entry && entry->textlen)
    {
        entry->textlen = 0;
        entry->text[0] = '\0';

        terminal_goto_xy (0, win->first_row + line_index);
        terminal_clear_to_eol ();
    }
}
```

In this case the value stored in `win->height` could be `-1`. This results in a very large unsigned number, causing invalid indexes and bad pointer dereferences to take place. Implementing the same hacked solution as before results in no more invalid indexing.

Another Crash:

```
* thread #1: tid = 0x9a72ee, 0x000000010bfbbca7 ginfo`display_update_on
frame #0: 0x000000010bfbbca7 ginfo`display_update_one_window(win=0x0000
    289
    290             /* This display line must both be in inverse, and h
    291             contents. */
 -> 292             if ((!display[line_index]->inverse) ||
    293                 (strcmp (display[line_index]->text, win->modeli
    294             {
    295                 terminal_goto_xy (0, line_index);
```

Again another NULL dereference to fix:

####Bug #6

Original:

```
if ((!display[line_index]->inverse) ||
    (strcmp (display[line_index]->text, win->modeline) != 0))
```

Patch:

```
if (display[line_index] != NULL && (display[line_index]->inverse == 0 |
    (strcmp (display[line_index]->text, win->modeline) != 0)))
```

These invalid indexing bugs are caused by the window height using an incorrect number. There are a lot of patches to find the cause of this.

####Bug #7

Original:

```
next->height--;
```

Patch:

```
if (next->height != 0) {
    next->height--;
}
```

Original:

```
prev->height--;
```

Patch:

```
if (prev->height != 0) {
    prev->height--;
}
```

Original:

```
active_window->height = the_screen->height - 1 - the_echo_area->height;
```

Patch:

```
active_window->height = the_screen->height - (the_screen->height > 2 ?
```

Original:

```
 if (win->height == delta_each)
    win->height -= (1 + the_echo_area->height);
```

Patch:

```
if (win->height == delta_each && win->height > 2)
    win->height -= (1 + the_echo_area->height);
```

Original:

```
if (win->height <= 0 || win->width <= 0 || display_inhibited)
    return;

```

Patch:

```
if (win->height <= 0 || win->height > INT32_MAX || win->width <= 0 || d
    return;

```

For final thoughts on this, the comment just above the past line of patched code reads:

```
/* If the window has no height, or display is inhibited, quit now.
Strictly speaking, it should only be necessary to test if the
values are equal to zero, since window_new_screen_size should
ensure that the window height/width never becomes negative, but
since historically this has often been the culprit for crashes, do
our best to be doubly safe.  */
```

At this point there is still a lingering a crash or two that I have yet to be able to trigger while info is attached in lldb. However now the mere act if resizing the window does not trigger a fatal crash immediately.

###Conclusion:

I don't feel like I have truly fixed anything from this. I have exposed a number of bugs related to poor string handling and made a poor attempt at holding back a tide of undefined behavior due to unindexed arrays. These bug fixes may actually cause more harm by being introduced than was caused by the original crashing bug. What started as a simple endeavor turned into an exercise in yak shaving and proof of murphy's law.

What made this all the more difficult wasn't the code syntax/formatting, or language, or even the age of the code itself. Lack of any architectural understanding was made this extremely difficult to fix. Most of these errors dealt with storing a negative value as the window height and resulting undefined behavior from that because it was using a unsigned integer to store a

signed integer value. The window height value is used to compute the number of stored lines of text to display, which might not be accurate to the actual number of indexed values in the array. No bounds-checking on this results in more problems due to accessing incorrect memory addresses and crashes. Simply put, many of these issues could have been avoided entirely by implementing safe coding practices in the first place.

If this blog post was helpful to you, please consider donating to keep this blog alive, thank you!

donate to support this blog