Erik Poole

I did static analysis on the bashbug portion of bash 3.2.

CASE 1: Uninitialized argument value

```
767   /* Initialize stack pointers.
768      Waste one element of value and location stack
769      so that they stay on the same level as the state stack.
770      The wasted elements are never initialized.  */
771
772   yyssp = yyss;
773   yyvsp = yyvs;
774 #if YYLSP_NEEDED
775   yylsp = yyls;
776 #endif
777   goto yysetstate;
      1 Control jumps to line 789 →
778
```

```
1043 case 3:
1044 #line 187 "plural.y"
1045 {
1046             yyval.exp = new_exp_2 (lor, yyvsp[-2].exp, yyvsp[0].exp);
                 16 ← 2nd function call argument is an uninitialized value
1047           }
1048     break;
```

The clang static analyzer here is warning that our bash program is dereferencing a pointer that never had the value at the other end of the pointer initialized. In most cases this will cause unexpected behavior since the value wasn't specified and won't be consistent across executions. However, in this case the pointer is being used to record the end of the stack and the pointer is being used to retrieve properly allocated data just beneath the stack pointer. In this case it doesn't matter what value the stack pointer is, just that it's in the right location relative to the other data in the stack. I don't believe this to be an error, and so It doesn't require a fix.

CASE 2: Dead Increment

```
314  main(argc, argv)
315       int argc;
316       char **argv;
317  {
```

```
344
345    argc -= optind;
       Value stored to 'argc' is never read
346    argv += optind;
347
```

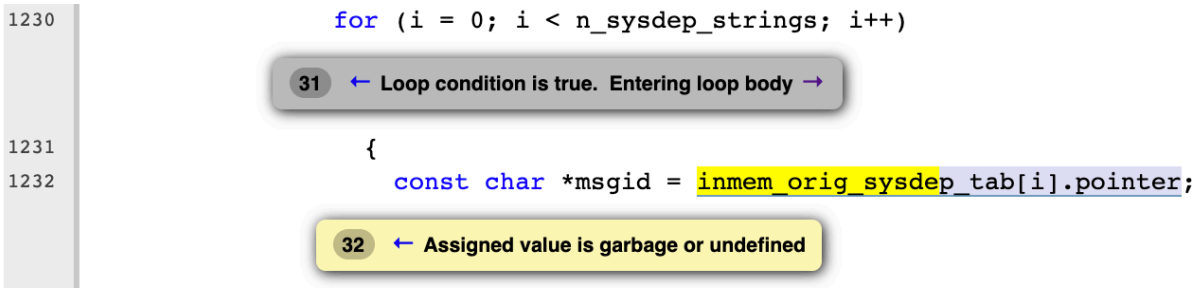
Both argv and argc are being incremented/decremented by an extern int “optind” but the internal values of argc and argv are never used again after being modified. This shouldn’t have any negative effect on the executable (and my indeed be compiled out, anyway) but also is unnecessary and should be removed. I would omit the lines.

CASE 3: Dead Assignment

```
348  if (filename)
349    {
350      fp = fopen (filename, "w");
351      if (fp == 0)
352        {
353          fprintf (stderr, "%s: %s: cannot open: %s\n", progname, filename, strerror(errno));
354          exit (1);
355        }
356    }
357  else
358    {
359      filename = "stdout";
       Value stored to 'filename' is never read
360      fp = stdout;
361    }
```

Similar to the last case, this assignment to “filename” is never used after this point. The logic makes some sense, if there isn’t a filename previously defined this code is setting it to a default value “stdout”, but this filename is never used after this assignment and should also be deleted.

CASE 4: Assigned Value is Garbage or Undefined



This statement is the end result of a very long if/else conditional tree. If all of the conditionals are false then it's possible that "inmem_orig_sysdep_tab" might not be initialized and the pointer values won't actually be pointing to anything of value. In that case the "msgid" pointer will be assigned to a garbage value. This is not a particularly easy fix, since the impression I get is that the original authors did not ever expect all of the conditional statements to fail. It might be reasonable to throw an error if the code ever reached this location and inmem_orig_sysdep_tab was still unassigned – since it's something the original authors maybe couldn't conceive of input that would cause it to occur.