

# Toybox Bug Analysis – infer

Austin Mordahl

July 18, 2018

## 1 Introduction

These bugs were generated by Infer v0.15.0 and Toybox 0.7.5. Bug reports are classified into the following categories:

True	A bug which exists and 1) its existence is unintended, or 2) whether or not its existence is purposeful is undetermined.
Technically True	A bug for which the content of the bug report is true, but whose existence is intended. The difference between a False and Technically True bug report is that the former could theoretically be detected by a more sophisticated implementation of the tool. Often, these bugs are “true but uninteresting.”
False	A bug the checker finds which, upon further inspection, does not exist in the code. For example, the checker indicating a variable is passed to a function without being initialized, when the variable is actually an out parameter and initialized within the function.

## 2 Statistics

---

True Reports	7
Technically True Reports	5
False Reports	68

---

## 3 Remarks

This analysis does not include bug reports having the bug type “Dead Store.” This type describes bugs in which a value is assigned to a variable but is never used. In the unique bug reports list, many bugs listed as unique are actually the same (for example, a variable infer recognizes as uninitialized will generate a unique bug report for every occurrence). For convenience, only one instance of each of these bugs is recorded here. [This is an issue I plan to resolve with updated versions of the scripts generating the unique bug reports].

## 4 True Reports

---

File	uptime.c
Line	54
Bug Type	NULL_DEREFERENCE
Description	pointer tm last assigned on line 53 could be null and is dereferenced at line 54, column 34
Number of Configurations	495

---

Code Sample	
53	tm = localtime(&t);
54	xprintf(" %02d:%02d:%02d up ", tm->tm_hour, tm->tm_min, tm->tm_sec);

---

Status	True
Remarks	localtime may fail and return a null pointer, causing tm to be null.

---

---

File	lib.c
Line	1268
Bug Class	MEMORY_LEAK
Description	gr is not reachable after line 1268, column 3.
Number of Configurations	986

---

#### Code Sample

```

1264 // Return group name or string representation of number, returned
      buffer
1265 // lasts until next call.
1266 char *getgroupname(gid_t gid)
1267 {
1268     struct group *gr = bufgetgrgid(gid);
1269     static char gnum[12];
1270
1271     sprintf(gnum, "%u", (unsigned)gid);
1272     return gr ? gr->gr_name : gnum;
1273 }

```

---

Status	True
Remarks	The call to bufgetgrgid initializes overwrites the list pointer with pointer to new memory without freeing the memory it pointed to initially. Although the buffer is meant to last past the end of the function, the memory is orphaned each time list is reassigned.

---

---

File	lib.c
Line	1257
Bug Class	MEMORY_LEAK
Description	pw is not reachable after line 1257, column 3.
Number of Configurations	986

---

Code Sample

```

1255 char *getusername(uid_t uid)
1256 {
1257     struct passwd *pw = bufgetpwuid(uid);
1258     static char unum[12];
1259
1260     sprintf(unum, "%u", (unsigned)uid);
1261     return pw ? pw->pw_name : unum;
1262 }

```

---

Status	True
Remarks	The same explanation for lib.c:1264 applies here.

---

---

File	mkflags.c
Line	126
Bug Type	NULL_DEREFERENCE
Description	pointer new last assigned on line 124 could be null and is dereferenced at line 126, column 7.
Number of Configurations	986

---

Code Sample

```

124     struct flag *new = calloc(sizeof(struct flag), 1);
125
126     new->command = string++;
127     new->next = list;
128     list = new;

```

---

Status	True
Remarks	<code>calloc</code> <i>can</i> fail, leaving new as a null pointer. Although this is true, it's not particularly interesting, as <code>calloc</code> very rarely fails.

---

---

File	mkflags.c
Line	101
Bug Type	NULL_DEREFERENCE
Description	pointer new last assigned on line 124 could be null and is dereferenced at line 126, column 7.
Number of Configurations	986

---

Code Sample

```

124     struct flag *new = calloc(sizeof(struct flag), 1);
125
126     new->command = string++;
127     new->next = list;
128     list = new;

```

---

Status	True
Remarks	calloc can fail, leaving new as a null pointer.
Similar Bugs	mkflags.c:227 – pointer new last assigned on line 90 could be null and is dereferenced at line 92, column 7

---

---

File	lib.c
Line	625
Bug Type	RESOURCE_LEAK
Description	resource acquired by call to <code>open()</code> at line 625, column 31 is not released after line 625, column 17.
Number of Configurations	986

---

#### Code Sample

```

611 void loopfiles_rw(char **argv, int flags, int permissions,
612     void (*function)(int fd, char *name))
613 {
614     int fd, failok = !(flags&WARN_ONLY);
615
616     flags &= ~WARN_ONLY;
617
618     // If no arguments, read from stdin.
619     if (!*argv) function((flags & O_ACCMODE) != O_RDONLY ? 1 : 0, "-");
620     else do {
621         // Filename "-" means read from stdin.
622         // Inability to open a file prints a warning, but doesn't exit.
623
624         if (!strcmp(*argv, "-")) fd = 0;
625         else if (0 > (fd = notstdio(open(*argv, flags, permissions))) && !
        failok) {
626             perror_msg_raw(*argv);
627             continue;
628         }
629         function(fd, *argv);
630         if ((flags & O_CLOEXEC) && fd) close(fd);
631     } while (++argv);
632 }

```

---

Status	True
Remarks	If <code>O_CLOEXEC</code> is passed to <code>loopfiles_rw</code> , then <code>fd</code> is closed. If <code>O_CLOEXEC</code> is not passed to <code>loopfiles_rw</code> , then whatever function is passed to <code>loopfiles_rw</code> must close <code>fd</code> itself. Of the three calls in <code>toybox</code> to <code>loopfiles_rw</code> which does not pass <code>O_CLOEXEC</code> , the call in <code>paste.c</code> does not close <code>fd</code> itself.

---



## 5 Technically True Reports

File	nohup.c
Line	27
Bug Type	RESOURCE_LEAK
Description	resource acquired by call to <code>open()</code> at line 27, column 15 is not released after line 27, column 9.
Number of Configurations	476

---

Code Sample

```

25  if (isatty(1)) {
26      close(1);
27      if (-1 == open("nohup.out", O_CREAT|O_APPEND|O_WRONLY,
28          S_IRUSR|S_IWUSR ))
29      {
30          char *temp = getenv("HOME");
31
32          temp = xprintf("%s/%s", temp ? temp : "", "nohup.out");
33          xcreate(temp, O_CREAT|O_APPEND|O_WRONLY, 0600);
34          free(temp);
35      }
36  }
```

---

Status	Technically True
Remarks	The file descriptor opened cannot be closed, as the whole point is to redirect <code>stdin</code> and <code>stdout</code> and then run a command elsewhere. This is not a bug, but at the same time infer is not wrong that there is an open file descriptor that is not being closed. Other bugs which involve open resources that are not closed but are meant to stay open are listed below.
Similar Bugs	<p><code>oneit.c:72</code> – resource acquired by call to <code>xopen_stdio()</code> at line 72, column 12 is not released after line 72, column 12.</p> <p><code>nohup.c:39</code> – resource acquired by call to <code>xopen_stdio()</code> at line 39, column 5 is not released after line 39, column 5.</p> <p><code>oneit.c:73</code> – resource acquired by call to <code>xopen_stdio()</code> at line 72, column 12 is not released after line 73, column 5.</p>

---

File	mountpoint.c:53
Bug Type	Memory Leak
Description	memory dynamically allocated by call to <code>xmprintf()</code> at line 51, column 9 is not reachable after line 53, column 7.
Number of Configurations	255
Code Sample	
<pre> 51  arg = xmprintf("%s/..", arg); 52  xstat(arg, &amp;st2); 53  if (CFG_TOYBOX_FREE) free(arg); </pre>	
Status	Technically True
Remarks	<p>From the Toybox configuration documentation (<code>Config.in</code>):</p> <p>“When a program exits, the operating system will clean up after it (free memory, close files, etc). To save size, toybox usually relies on this behavior. If you’re running toybox under a debugger or without a real OS (ala newlib+libgloss), enable this to make toybox clean up after itself.” If <code>TOYBOX_FREE</code> is not enabled (and by default, it isn’t), the memory allocated by <code>xmprintf</code> will not be freed explicitly, and instead will be left open until the program terminates so the operating system can clean it up. This makes classifying this bug tricky. This is obviously a memory leak in the default case, as memory that is being allocated is not being freed; however, it seems that this is an intentional choice by the programmers. For now, this seems to fall nicely under the technically true class, but this could warrant a discussion on how exactly we’re classifying what a bug is.</p>

## 6 False Reports

File	grep.c
Line	184
Bug Type	UNINITIALIZED_VALUE
Description	The value read from <code>matches.rm_so</code> was never initialized.
Number of Configurations	507

---

Code Sample	
178	<code>if (toys.optflags &amp; FLAG_v) {</code>
179	<code>if (toys.optflags &amp; FLAG_o) {</code>
180	<code>if (rc) skip = matches.rm_eo = strlen(start);</code>
181	<code>else if (!matches.rm_so) {</code>
182	<code>start += skip;</code>
183	<code>continue;</code>
184	<code>} else matches.rm_eo = matches.rm_so;</code>
185	<code>} else {</code>
186	<code>if (!rc) break;</code>
187	<code>matches.rm_eo = strlen(start);</code>
188	<code>}</code>
189	<code>matches.rm_so = 0;</code>
190	<code>} else if (rc) break;</code>

---

Status	False
Remarks	Were <code>matches.rm_so</code> a singular variable, infer would be correct, because the initialization of <code>matches.rm_so</code> would be out of scope. However, <code>matches</code> is a struct which is in scope. Additionally, the <code>else if</code> clause checks whether <code>matches.rm_so</code> exists; line 184 will not be reached if <code>matches.rm_so</code> is not initialized.

---

---

File	xwrap.c
Line	389
Bug Type	RESOURCE_LEAK
Description	resource acquired by call to <code>xopen_stdio()</code> at line 389, column 19 is not released after line 389, column 3.
Number of Configurations	986

---

#### Code Sample

```

330 int xcreate_stdio(char *path, int flags, int mode)
331 {
332     int fd = open(path, (flags^O_CLOEXEC)&~WARN_ONLY, mode);
333
334     if (fd == -1) ((mode&WARN_ONLY) ? perror_msg_raw : perror_exit_raw)(
        path);
335     return fd;
336 }
337
338 // Die unless we can open a file, returning file descriptor.
339 int xopen_stdio(char *path, int flags)
340 {
341     return xcreate_stdio(path, flags, 0);
342 }

```

---

Status	False
Remarks	<code>xopen_stdio()</code> automatically closes a file unless the <code>O_CLOEXEC</code> flag is passed to it (behaves opposite other functions which open files). There are two calls to <code>xopen_stdio()</code> which do not pass <code>O_CLOEXEC</code> . The first is in <code>oneit.c</code> , line 99. Here, the file descriptors are kept open on purpose, redirecting <code>stdin</code> , <code>stdout</code> , and <code>stderr</code> . The same pattern is used in the second occurrence. in <code>getty.c</code> .

---

File	xwrap.c
Line	458
Bug Type	NULL_DEREFERENCE
Description	pointer null could be null and is dereferenced by call to <code>getcwd()</code> at line 458, column 15.
Number of Configurations	986

---

#### Code Sample

```

456 char *xgetcwd(void)
457 {
458     char *buf = getcwd(NULL, 0);
459     if (!buf) perror_exit("xgetcwd");
460
461     return buf;
462 }

```

---

Status	False
Remarks	The usage of the NULL pointer should not trigger this error.

---

#### Similar Bugs

*Condition:* Similar bug reports are those in which infer incorrectly identifies a NULL pointer as a Null Dereference bug.

`pwd.c:26` pointer null could be null and is dereferenced by call to `getcwd()` at line 26, column 19.

---

---

	File	xwrap.c
	Line	264
Bug Type	UNINITIALIZED_VALUE	
	Description	The value read from cestnepasun[-] was never initialized.
	Number of Configurations	986

---

#### Code Sample

```

262  if (pipes) {
263      if (pipes[0] != -1) close(cestnepasun[0]);
264      if (pipes[1] != -1) close(cestnepasun[3]);
265  }

```

---

Status	False
Remarks	<p>cestnepasun is initialized by a call to pipe. This is representative of a broader class of bugs falling into the UNINITIALIZED_VALUE type. These bugs take the form of two separate branches with the same condition, wherein a variable is intialized in the first and used in the second (i.e. <code>int a; if (b) a = 5; ...; if (b) int c = a;</code> Other false bugs which fall into this class (and similar bugs, i.e. wherein infer doesn't understand the variable being initialized in a different scope) are listed below.</p>
Similar Bugs	<p>xwrap.c:287 – The value read from pipe was never initialized.</p> <p>mount.c:314 – The value read from mtl was never initialized.</p> <p>comm.c:79 – The value read from file[_] was never intialized.</p> <p>kill.c:145 – The value read from signum was never intialized.</p> <p>xwrap.c:790 – The value read from d was never intialized.</p> <p>xwrap.c:737 – The value read from fd was never initialized.</p> <p>bzcat.c:252 – The value read from length[_] was never initialized.</p> <p>modinfo.c:496 – The value readf from len was never initialized.</p> <p>ps.c:1514 – The value read from lines was never initialized.</p> <p>md5sum.c:203 – The value read from rot[_] was never initialized.</p> <p>setenforce.c:30 – The value read from state was never initialized.</p> <p>od.c:92 – The value read from ld was never initialized.</p> <p>xwrap.c:794 – The value read from l was never initialized.</p> <p>hwclock.c:466 – The value read from time was never initialized.</p>

---

---

File	ulimit.c
Line	95
Bug Type	UNINITIALIZED_VALUE
Description	The value read from <code>rr.rlim_cur</code> was never initialized.
Number of Configurations	510

---

Code Sample

```

84     if ((l<<i)&FLAG_p) {
85         if (toys.optflags&FLAG_H)
86             xreadfile("/proc/sys/fs/pipe-max-size", toybuf, sizeof(
toybuf));
87     else {
88         int pp[2];
89
90         xpipe(pp);
91         sprintf(toybuf, "%d\n", fcntl(*pp, F_GETPIPE_SZ));
92     }
93     printf("%s", toybuf);
94 } else {
95     rlim_t rl = (toys.optflags&FLAG_H) ? rr.rlim_max : rr.
rlim_cur;
96
97     if (rl == RLIM_INFINITY) printf("unlimited\n");
98     else printf("%ld\n", (long)rl);
99 }

```

---

Status	False
Remarks	<code>rr</code> is initialized by a call to <code>prlimit</code> . This bug is representative of a broader class of bugs falling into the <code>UNINITIALIZED_VALUE</code> type. These bugs take the form of a struct being initialized and then a field in that struct being referenced later. Other false bugs which fall into this class are listed below.
Similar Bugs	<code>od.c:88</code> – The value read from <code>fdl.ld</code> was never initialized. <code>od.c:85</code> – The value read from <code>fdl.d</code> was never initialized. <code>ulimit.c:95</code> – The value read from <code>rr.rlim_max</code> was never initialized. <code>od.c:491</code> – The value read from <code>fdl.f</code> was never initialized. <code>grep.c:215</code> – The value read from <code>matches.rm_so</code> was never initialized.

---



File	ls.c
Line	428
Bug Type	UNINITIALIZED_VALUE
Description	The value read from totals[-] was never initialized.
Number of Configurations	655

---

#### Code Sample

```

428     if (flags & FLAG_s) {
429         print_with_h(tmp, st->st_blocks, 512);
430         printf("%*s ", totals[6], tmp);
431     }

```

---

Status	False
Remarks	totals was initialized by memset. Other bugs in which the variable was initialized by memset, memcpy, or any other function which accepts out parameters are listed below.
Similar Bugs	md5sum.c:157 – The value read from x[-] was never initialized.
Similar Bugs	ps.c:1453 – The value read from run[-] was never initialized.
	md5sum.c:143 – The value read from x[-] was never initialized.
	ftpget.c:150 – The value read from port was never initialized.
	ftpget.c:151 – The value read from si6.sin6.family was never initialized.
	ls.c:377 – The value read from dtlen was never initialized.
	rfkill.c:94 – The value read from rfevent.idx was never initialized.
	ls.c:479 – The value read from totals[-] was never initialized.
	ls.c:349 – The value read from dtlen was never initialized.

---

---

File	pmap.c
Line	86
Bug Type	UNINITIALIZED_VALUE
Description	The value read from swap was never intialized.
Number of Configurations	500

---

Code Sample

```

78     if (0<sscanf(line, "Pss: %lld", &pss)
79         || 0<sscanf(line, "Private_Dirty: %lld", &dirty)
80         || 0<sscanf(line, "Swap: %lld", &swap)) xx++;
81     free(line);
82     if (xx<3) continue;
83     line = oldline;
84     name = basename(name);
85     xx = 0;
86     printf("%7lld %7lld %7lld ", pss, dirty, swap);

```

---

Status	False
Remarks	swap is initialized in the loop above.
Similar Bugs	pmap.c:86 – The value read from dirty was never initialized.

---

File	sed.c
Line	671
Bug Type	MEMORY_LEAK
Description	Memory dynamically allocated to <code>return</code> by call to <code>xmalloc()</code> at line 668, column 16 is not reachable after line 671, column 17.
Number of Configurations	494
Status	False
Remarks	There is no <code>return</code> variable. (This is the same bug as <code>sed.c:694</code> )

File	xwrap.c
Line	383
Bug Type	RESOURCE_LEAK
Description	resource acquired by call to xcreate_stdio() at line 383, column 19 is not released after line 383, column 3.
Number of Configurations	986
Status	False
Remarks	No calls to xcreate pass O_CLOEXEC, so the file will always be closed automatically.
Similar Bugs	
<i>Condition</i>	xopen and xcreate close files by default unless O_CLOEXEC is passed to them. These bugs involve instances in which the file opened by those functions is not closed explicitly, but O_CLOEXEC is not passed; therefore, we can safely assume that the file is closed.
xwrap.c:373	resource acquired by call to xopen_stdio() at line 373, column 5 is not released after line 373, column 5.

File	password.c:986
Bug Type	Null Dereference
Description	pointer <code>sfx</code> last assigned on line 119 could be null and is dereferenced at line 127, column 3.
Number of Configurations	986
Status	False
Remarks	The string <code>strchr</code> searches is guaranteed to have a '+' in it by <code>xmprintf</code> , so <code>strchr</code> will never fail and <code>sfx</code> will never be null.

File	lsm.h:63
Bug Type	Uninitialized Value
Description	The value read from <code>result</code> was never initialized.
Number of Configurations	922
Status	False
Remarks	See the report from <code>cppcheck</code> for the same bug.