Toybox Bug Analysis – infer

Austin Mordahl

July 16, 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 cppcheck 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 cppcheck.
False	A bug cppcheck finds which, upon further inspection, does not exist in the code. For example, cppcheck indicating a variable is passed to a function without being initialized, when the variable is actually an out parameter and intialized within the function.

2 True Reports

3 Technically True Reports

4 False Reports

File	grep.c
Line	184
Description	The value read from `matches.rm_so` was never initialized.
Number of Configurations	507

Code Sample

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

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

File	xwrap.c
Line	389
Description	resource acquired by call to `xopen_stdio()` at line 389, column 19 is not released after line 389, column 3.
Number of Configurations	986

Code Sample

```c
int xcreate_stdio(char *path, int flags, int mode)
{
  int fd = open(path, (flags^O_CLOEXEC)&~WARN_ONLY, mode);

  if (fd == -1) ((mode&WARN_ONLY) ? perror_msg_raw : perror_exit_raw)(
    path);
  return fd;
}

// Die unless we can open a file, returning file descriptor.
int xopen_stdio(char *path, int flags)
{
  return xcreate_stdio(path, flags, 0);
}
```

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