Static Analysis and Formal Verification at Red Hat

Kamil Dudka <kdudka@redhat.com>

September 11th 2019

Abstract

Red Hat uses static analyzers to automatically find bugs in the source code of Red Hat Enterprise Linux. The distribution consists of approx. 3000 RPM packages and 300 million lines of code. Red Hat develops an open source tool that can statically analyze this amount of software in a fully automatic way. We give it source RPM packages of our choice and get the results of selected static analyzers in a unified machine-readable format. This talk will cover which static analyzers are used by Red Hat and how their results are handled. Red Hat is now also experimenting with formal verifiers Symbiotic and Divine, which are developed by research groups of Masaryk University. Is there any chance to integrate such tools into Red Hat's static analysis workflow?

Red Hat

Why do we use static analysis at Red Hat?

- ... to find programming mistakes soon enough – example:

```
Error: SHELLCHECK_WARNING:
/etc/rc.d/init.d/squid:136:10: warning: Use "${var:?}" to ensure this never expands to /* .
#  134|         RETVAL=$?
#  135|         if [ $RETVAL -eq 0 ] ; then
#  136|->               rm -rf $SQUID_PIDFILE_DIR/*
#  137|               start
#  138|         else
```

https://bugzilla.redhat.com/1202858 – *[UNRELEASED] restarting testing build of squid results in deleting all files in hard-drive*

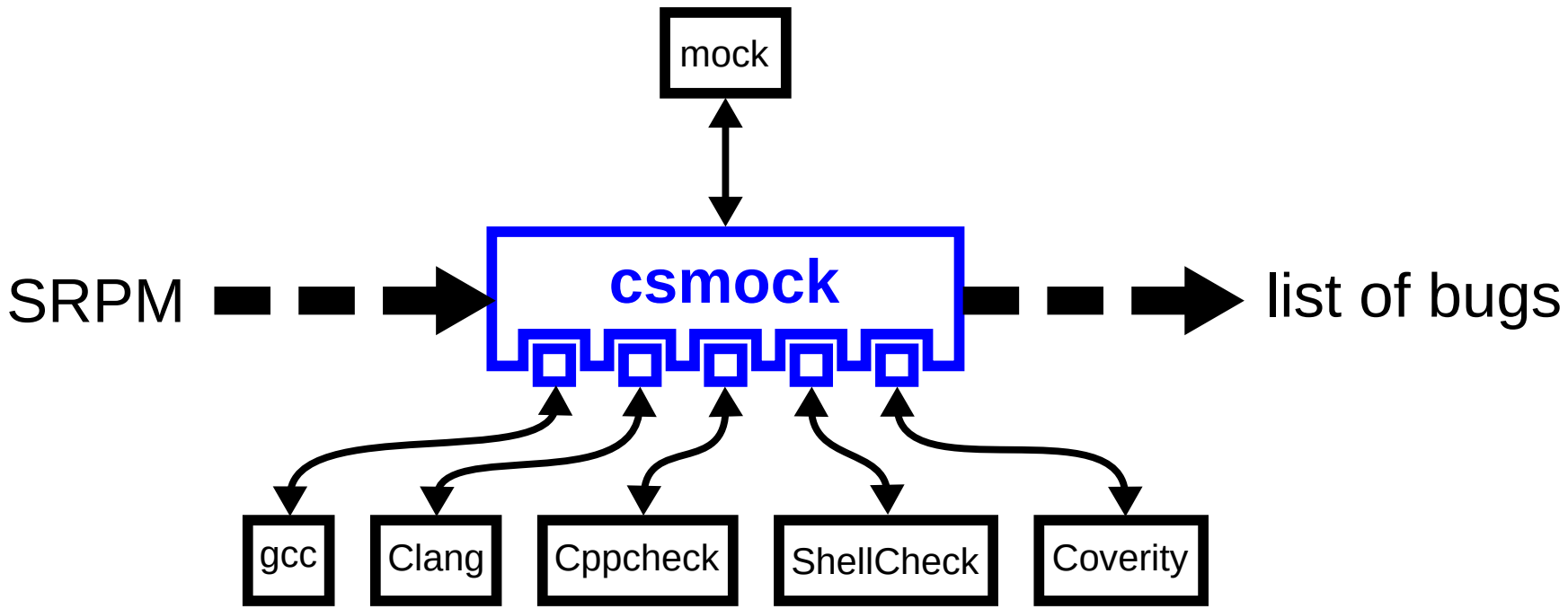
- Static analysis is required for Common Criteria certification.

Static Analysis at Red Hat in Numbers

- RHEL-8 Beta static analysis mass scan in July 2018
- analyzed 318 million LoC (Lines of Code) in 3390 packages
- 95% packages scanned successfully
- approx. 370 000 potential bugs detected in total
- approx. one potential bug per 1000 LoC

csmock

- command-line tool that runs static analyzers
- one interface, one output format, plug-in API
- fully open-source, available in Fedora/CentOS



csmock – Supported Static Analyzers

	C	C++	Java	Go	JavaScript	PHP	Python	Ruby	Shell
gcc	✓	✓							
Clang	✓	✓							
Cppcheck	✓	✓							
Coverity	✓	✓	✓	✓	✓	✓	✓	✓	
ShellCheck									✓
Pylint							✓		
Bandit							✓		
Smatch	✓								

Need more?

https://github.com/mre/awesome-static-analysis#user-content-programming-languages-1

What is important for developers?

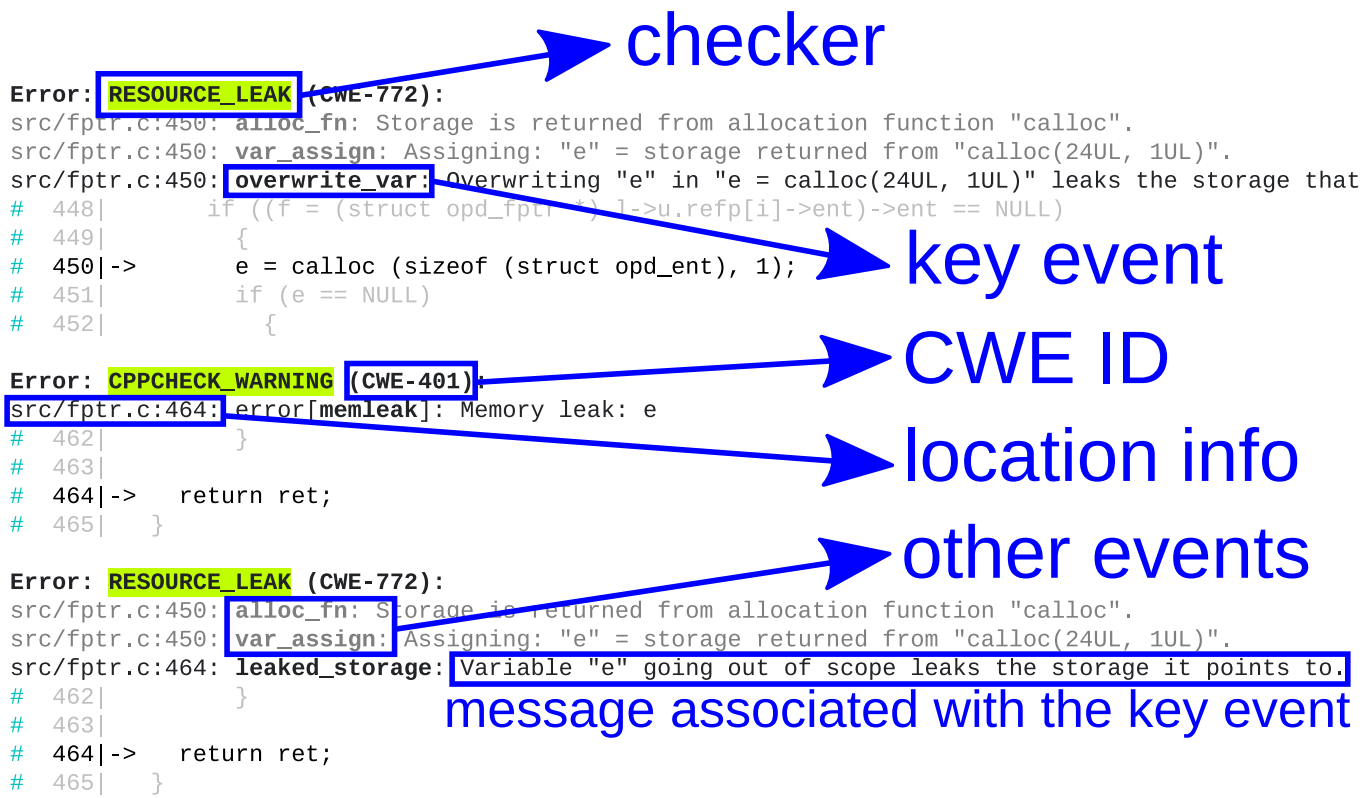
The static analyzers need to:

- be fully automatic
- provide reasonable signal to noise ratio
- provide reproducible and consistent results
- be approximately as fast as compilation of the package
- support differential scans:
  - added/fixed bugs in an update?
  - https://github.com/kdudka/csdiff

csmock – Output Format

```
Error: RESOURCE_LEAK (CWE-772):
src/fptr.c:450: alloc_fn: Storage is returned from allocation function "calloc".
src/fptr.c:450: var_assign: Assigning: "e" = storage returned from "calloc(24UL, 1UL)".
src/fptr.c:450: overwrite_var: Overwriting "e" in "e = calloc(24UL, 1UL)" leaks the storage that "e" points to.
#  448|       if ((f = (struct opd_fptr *) l->u.refp[i]->ent)->ent == NULL)
#  449|         {
#  450|->       e = calloc (sizeof (struct opd_ent), 1);
#  451|         if (e == NULL)
#  452|           {

Error: CPPCHECK_WARNING (CWE-401):
src/fptr.c:464: error[memleak]: Memory leak: e
#  462|         }
#  463|
#  464|->   return ret;
#  465|   }

Error: RESOURCE_LEAK (CWE-772):
src/fptr.c:450: alloc_fn: Storage is returned from allocation function "calloc".
src/fptr.c:450: var_assign: Assigning: "e" = storage returned from "calloc(24UL, 1UL)".
src/fptr.c:464: leaked_storage: Variable "e" going out of scope leaks the storage it points to.
#  462|         }
#  463|
#  464|->   return ret;
#  465|   }
```

csmock – Output Format




```
Error: RESOURCE_LEAK (CWE-772):
src/fptr.c:450: alloc_fn: Storage is returned from allocation function "calloc".
src/fptr.c:450: var_assign: Assigning: "e" = storage returned from "calloc(24UL, 1UL)".
src/fptr.c:450: overwrite_var: Overwriting "e" in "e = calloc(24UL, 1UL)" leaks the storage that "e" points to.
#  448|       if ((f = (struct opd_fptr *) l->u.refp[i]->ent)->ent == NULL)
#  449|         {
#  450|->       e = calloc (sizeof (struct opd_ent), 1);
#  451|         if (e == NULL)
#  452|           {

Error: CPPCHECK_WARNING (CWE-401):
src/fptr.c:464: error[memleak]: Memory leak: e
#  462|         }
#  463|
#  464|->   return ret;
#  465|   }

Error: RESOURCE_LEAK (CWE-772):
src/fptr.c:450: alloc_fn: Storage is returned from allocation function "calloc".
src/fptr.c:450: var_assign: Assigning: "e" = storage returned from "calloc(24UL, 1UL)".
src/fptr.c:464: leaked_storage: Variable "e" going out of scope leaks the storage it points to.
#  462|         }
#  463|
#  464|->   return ret;
#  465|   }
```

csmock – Output Format (Trace Events)

```
Error: RESOURCE_LEAK (CWE-772):
src/fptr.c:447: cond_true: Condition "i < l->nrefs", taking true branch.
src/fptr.c:448: cond_true: Condition "(f = (struct opd_fptr *)l->u.refp[i]->ent)->ent == NULL", taking true branch.
src/fptr.c:450: alloc_fn: Storage is returned from allocation function "calloc".
src/fptr.c:450: var_assign: Assigning: "e" = storage returned from "calloc(24UL, 1UL)".
src/fptr.c:451: cond_false: Condition "e == NULL", taking false branch.
src/fptr.c:456: if_end: End of if statement.
src/fptr.c:462: loop: Jumping back to the beginning of the loop.
src/fptr.c:447: loop_begin: Jumped back to beginning of loop.
src/fptr.c:447: cond_true: Condition "i < l->nrefs", taking true branch.
src/fptr.c:448: cond_true: Condition "(f = (struct opd_fptr *)l->u.refp[i]->ent)->ent == NULL", taking true branch.
src/fptr.c:450: overwrite_var: Overwriting "e" in "e = calloc(24UL, 1UL)" leaks the storage that "e" points to.
#  448|       if ((f = (struct opd_fptr *) l->u.refp[i]->ent)->ent == NULL)
#  449|         {
#  450|->       e = calloc (sizeof (struct opd_ent), 1);
#  451|         if (e == NULL)
#  452|           {
```

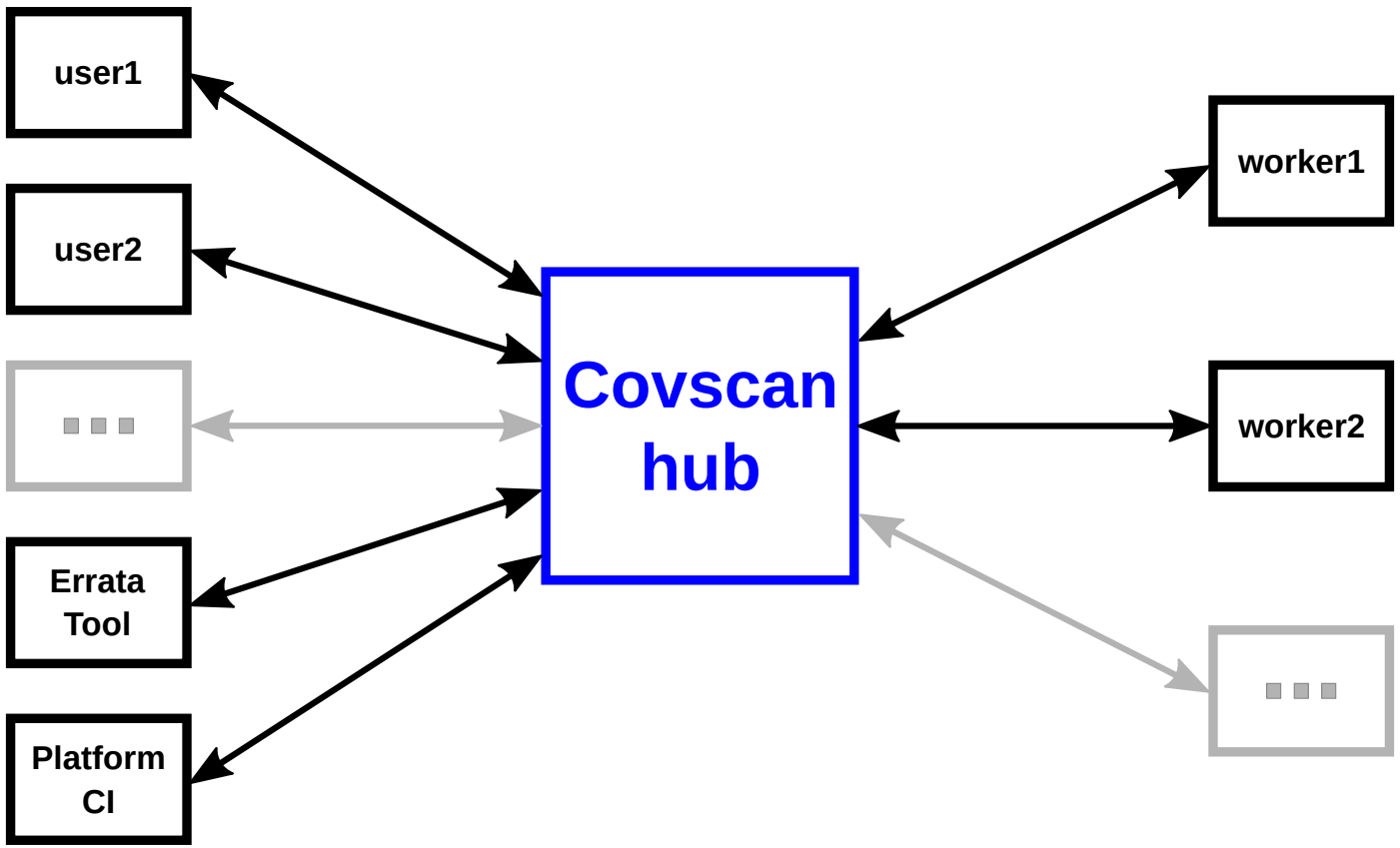
Red Hat

Example of a Fix

```
--- a/src/fptr.c
+++ b/src/fptr.c
@@ -438,28 +438,29 @@
 GElf_Addr
 opd_size (struct prelink_info *info, GElf_Word entsize)
 {
   struct opd_lib *l = info->ent->opd;
   int i;
   GElf_Addr ret = 0;
   struct opd_ent *e;
   struct opd_fptr *f;

   for (i = 0; i < l->nrefs; ++i)
     if ((f = (struct opd_fptr *) l->u.refp[i]->ent)->ent == NULL)
       {
        e = calloc (sizeof (struct opd_ent), 1);
        if (e == NULL)
          {
            error (0, ENOMEM, "%s: Could not create OPD table",
                   info->ent->filename);
            return -1;
          }

        e->val = f->val;
        e->gp = f->gp;
        e->opd = ret | OPD_ENT_NEW;
+       f->ent = e;
        ret += entsize;
       }

   return ret;
 }
```

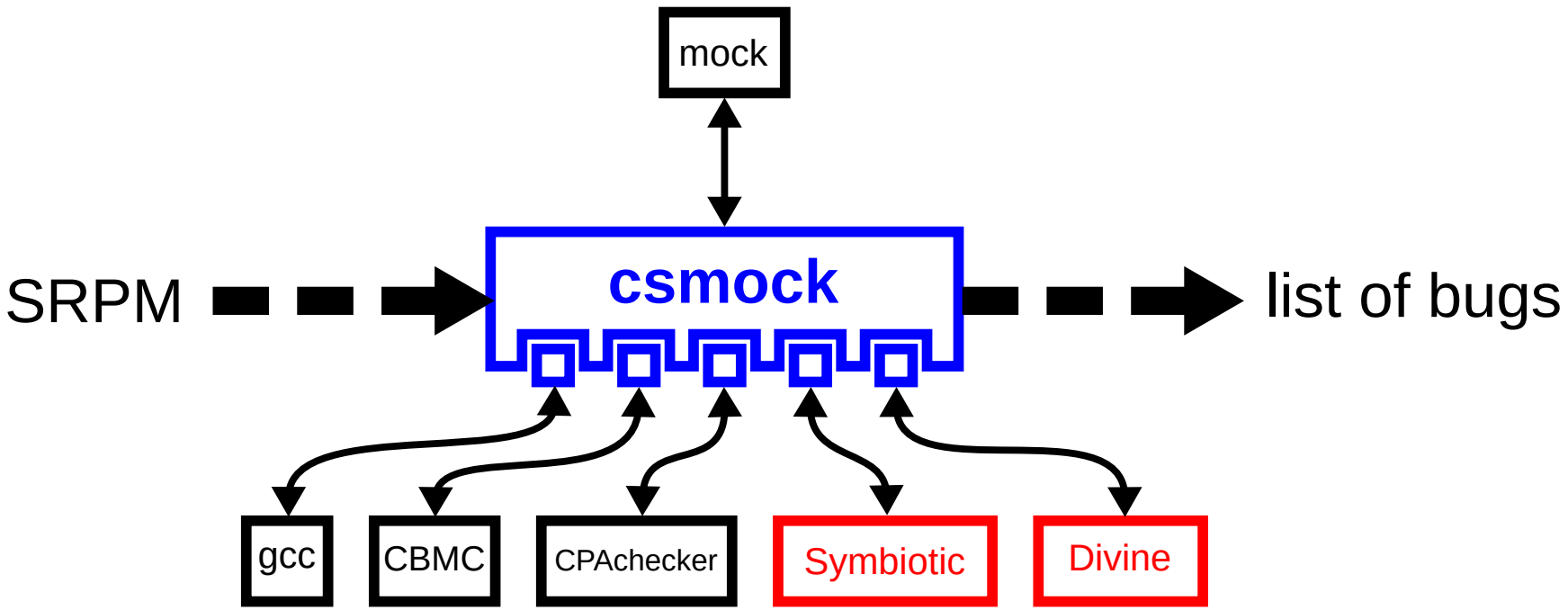
Covscan

- Red Hat's internal service that runs csmock.



Red Hat

Integration of Formal Verifiers – Goal



Integration of Formal Verifiers – Reality

- Problems:
  - Our developers fail to compile formal verifiers.
  - Formal verifiers fail to compile our source code.
  - How to deal with missing models of external functions?
  - RPM packages have `0..n` definitions of `main()`.
  - Problems with scalability have not yet been reached.
- Solutions:
  - Symbiotic and Divine are now available as RPM packages.
  - Working on support for dynamic analyzers in csmock (for RPMs that run test-suite during the build).