Formal Verification of a Linux Distribution

Kamil Dudka <kdudka@redhat.com>

January 23rd 2020

Abstract

Red Hat uses static analyzers to automatically find bugs in the source code of Red Hat Enterprise Linux, consisting of approx. 3000 RPM packages and 300 million lines of code. There are open source tools that can statically analyze this amount of software in a fully automatic way. Would it be possible to use formal verification tools to find bugs in (or even prove correctness of) the important pieces of code in our Linux distribution? Red Hat is now experimenting with formal verifiers Symbiotic and Divine, which are developed by research groups of Masaryk University in Brno. Are these tools ready for industrial software? How much are we able to integrate them into our release pipeline?

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 of a Linux Distribution

- Static analysis of Red Hat Enterprise Linux (RHEL):
  - approx. 300 000 000 lines of code
  - approx. 370 000 potential bugs
  - approx. 3 000 RPM packages
- Full scan of each major release of RHEL.
- Differential scans of subsequent updates.
- Developers are automatically notified about potential bugs.

Red Hat

Is it sufficient to use static analyzers?

- Static analyzers are really fast and flexible.
- But they usually suffer from limited precision.
- False positives and false negatives are expected.
- This causes human resources to be wasted:
  - when reviewing numerous false positives
  - when updating already released products
    (due to bugs not revealed soon enough).

Red Hat

Let’s use formal verification!

Formal verification tools (a.k.a. formal verifiers):

- can guarantee zero false positives/negatives.
- can prove correctness of programs.
- are developed at Masaryk University:
  - Divine – explicit-state model checking
  - Symbiotic – instrumentation, slicing and symbolic execution
- attend Competition on Software Verification (SV-COMP):
  - `https://sv-comp.sosy-lab.org/`

Challenges with Formal Verifiers

On the other hand, formal verifiers:

- fail to compile (or understand) our code.
- are much more hungry on computational resources.
- do not finish in predictable amount of time.
- do not like libraries and external functions.
- usually expect small isolated programs.

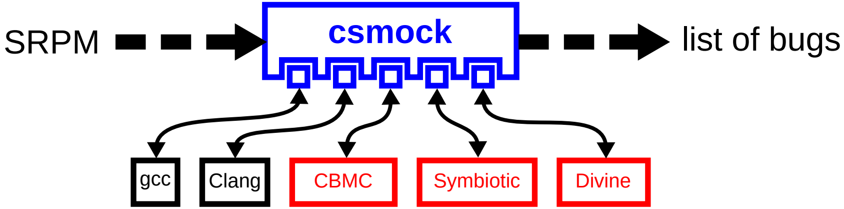
Integration of Formal Verifiers

- Symbiotic and Divine are now available as RPM packages:
  https://copr.fedorainfracloud.org/coprs/jamartis/symbiotic/
  https://copr.fedorainfracloud.org/coprs/lzaoral/Divine/
- Symbiotic was extended to report multiple bugs in one run.
- Started to experiment with CBMC for independent comparison.
- Developing output converters for Divine, Symbiotic and CBMC:

```
Error: SYMBIOTIC_WARNING:
test-0131.c:122: error: memory error: out of bound pointer
test-0131.c:143: note: from call of traverse()
test-0131.c:127: note: from call of main()
```

Integration of Formal Verifiers – csmock

- Command-line tool to run (not only) static analyzers.
- One interface, one output format, plug-in API.
- Fully open-source, available in Fedora/CentOS.


SRPM
csmock
list of bugs
gcc
Clang
CBMC
Symbiotic
Divine