Static Analysis of a Linux Distribution

Kamil Dudka <kdudka@redhat.com>

October 7th 2019

How to find programming mistakes efficiently?

0. users (preferably volunteers)
1. Automatic Bug Reporting Tool (ABRT)
2. code review, automated tests, dynamic analysis
3. static analysis!

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.

Agenda

1. Code Review, Dynamic Analysis, Fuzzing
2. Linux Distribution, Reproducible Builds
3. Static Analysis of a Linux Distribution
4. Formal Verification

Red Hat

Code Review

- design (anti-)patterns
- error handling (OOM, permission denied, . . .)
- validation of input data (headers, length, encoding, . . .)
- sensitive data treatment (avoid exposing private keys, . . .)
- use of crypto algorithms
- resource management

Red Hat

Dynamic Analysis

- good to have some test-suite to begin with
- memory error detectors, profilers, e.g. valgrind
- tools to measure test coverage, e.g. gcov/lcov
- compiler instrumentation, e.g. GCC built-in sanitizers (address sanitizer, thread sanitizer, UB sanitizer, . . .)
- not so easy to automate as static analysis

Red Hat

Fuzzing

- feeding programs with unusual input
- can be combined with valgrind, GCC sanitizers, etc.
- `radamsa` – general purpose data fuzzer

```
$ cat file | radamsa | program
```

- OSS-Fuzz – continuous fuzzing of open source software
  - service provided by Google
  - many security issues detected e.g. in curl

Agenda

1. Code Review, Dynamic Analysis, Fuzzing
2. **Linux Distribution, Reproducible Builds**
3. Static Analysis of a Linux Distribution
4. Formal Verification

Linux Distribution

- operating system (OS)
- based on the Linux kernel
- a lot of other programs running in user space
- usually open source

Red Hat

Upstream vs. Downstream

- upstream SW projects – usually independent
- downstream distribution of upstream SW projects
  - Red Hat uses the RPM package manager
  - files on the file system owned by packages:
    - dependencies form an oriented graph over packages
    - we can query package database
    - we can verify installed packages

rpm

Red Hat

Fedora vs. RHEL

- Fedora
  - new features available early
  - driven by the community (developers, users, . . .)
- RHEL (Red Hat Enterprise Linux)
  - stability and security of existing deployments
  - driven by Red Hat (and its customers)

Red Hat

Where do RPM packages come from?

- developers maintain source RPM packages (SRPMs)
- binary RPMs can be built from SRPMs using `rpmbuild`:

```
rpmbuild --rebuild git-2.6.3-1.fc24.src.rpm
```

- binary RPMs can be then installed on the system:

```
sudo dnf install git
```

Red Hat

Reproducible Builds

- local builds are not reproducible
- `mock` – chroot-based tool for building RPMs:

```
mock -r fedora-rawhide-i386 git-2.6.3-1.fc24.src.rpm
```

- `koji` – service for scheduling build tasks

```
koji build rawhide git-2.6.3-1.fc24.src.rpm
```

- easy to hook static analyzers on the build process!

Red Hat

Reproducible Builds – Obstacles

- build env not 100% isolated from host env
- toolchain (compiler, linker, glibc, . . .) evolves
- parallel builds with missing dependencies (tricky to debug)
- installation of binary RPMs not (always) reproducible
- too many unexpected side effects – examples:
  - SMTP server fails to build on up2date kernel
  - one-line change of a man page doubles size of curl binary
  - cookies and certificates in curl upstream test-suite expire
  - autoconf tests: https://github.com/curl/curl/commit/curl-7_49_1-45-gb2dcf0347

Red Hat

Reproducible Builds – Best Practices

- use `git archive` to create tarballs
  (does not work well with autotools)
- isolate build env from host env
  (chroot, mock, containers, VMs)
- do not use compiler flags like `-mtune=native`
- disable Internet acess during the build
- sign release tags and release tarballs

Agenda

1 Code Review, Dynamic Analysis, Fuzzing

2 Linux Distribution, Reproducible Builds

3 **Static Analysis of a Linux Distribution**

4 Formal Verification

Red Hat

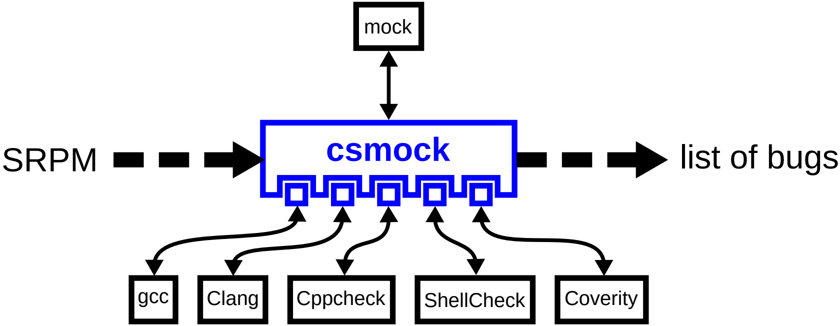
Static Analysis at Red Hat in Numbers

- 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

Red Hat

csmock

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



Red Hat

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

Red Hat

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

Red Hat

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|   }
```

Red Hat

csmock – Output Format

checker

```
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 *) )->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|   }
```

key event

CWE ID

location info

other events

message associated with the key event

Red Hat

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;
 }
```

Red Hat

Example – Differential Scan of logrotate (1/2)

- Someone opened a pull request for logrotate:
  https://github.com/logrotate/logrotate/pull/146:

  ```
  logrotate.c:251:15: warning: Result of 'malloc' is
  converted to a pointer of type 'struct logStates',
  which is incompatible with sizeof operand type
  'struct logState'
  ```

- Next day we agreed on a fix and pushed it:
  https://github.com/logrotate/logrotate/pull/149

Red Hat

Example – Differential Scan of logrotate (2/2)

- One day before the release I ran a differential scan with the csbuild utility – demo:

```
git clone https://github.com/logrotate/logrotate.git
cd logrotate && git reset --hard eb322705^
autoreconf -fiv && ./configure
BUILD_CMD='make clean && make -j9'
csbuild -c $BUILD_CMD -g 3.12.3..master --git-bisect
```

- Luckily, I was able to fix it properly before the release: https://github.com/logrotate/logrotate/commit/eb322705

```
csbuild -c $BUILD_CMD -g origin..master --print-fixed
```

Red Hat

Upstream vs. Enterprise

Different approaches to static analysis:

upstream – fix as many bugs as possible

- false positive ratio increases over time!

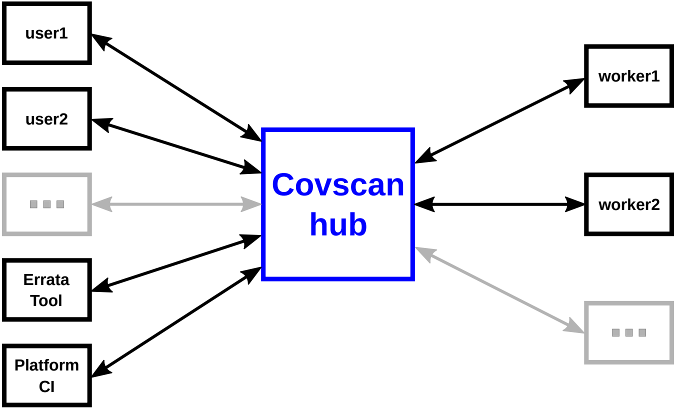
enterprise – run differential scans to verify code changes

- up to 10% of bugs usually detected as new in an update
- up to 10% of them usually confirmed as real by developers

Red Hat

Covscan

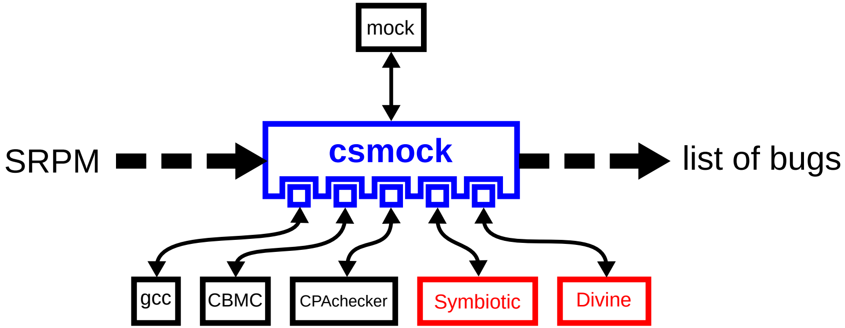
- Red Hat's internal service that runs csmock.



Agenda

Red Hat

Integration of Formal Verifiers – Goal



Need more?

https://sv-comp.sosy-lab.org/2019/results/results-verified/

Red Hat

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).

Slides Available Online

https://kdudka.fedorapeople.org/muni19.pdf