Static and Dynamic Analysis of a Linux Distribution

Red Hat
Vincent Mihalkovič, Lukáš Zaoral

8th December 2023

Why do we use code 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.

Red Hat

Agenda

1 Linux Distribution, Reproducible Builds

2 Static Analysis of a Linux Distribution

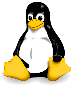
3 Dynamic Analysis of a Linux Distribution

4 OpenScanHub

5 Differential ShellCheck

What is a 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 RPM packages.

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-2.39.2-1.fc39.src.rpm
```

- Binary RPMs can be then installed on the system:

```
sudo dnf install git
```

Reproducible Builds

- Local builds are not easily reproducible.
- mock – container-based tool for building RPMs:

```
mock -r fedora-rawhide-x86_64 git-2.43.0-1.fc40.src.rpm
```

- Easy to hook static analyzers into the build process!
- Who cares about reproducible builds?
  https://reproducible-builds.org/who/projects/

Agenda

1 Linux Distribution, Reproducible Builds

2 **Static Analysis of a Linux Distribution**

3 Dynamic Analysis of a Linux Distribution

4 OpenScanHub

5 Differential ShellCheck

Static Analysis of a Linux Distribution

- Vast range of software packages, each developed independently and with various contributors.
- Huge number of (potential?) defects in certain projects.
- No control over technologies and programming languages.
- No control over upstream coding style.
- It is impossible for a single person to be familiar with all the code of a large project.

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 are usually detected as new in an update.
  - Up to 10% of them are usually confirmed as real by developers.

Red Hat

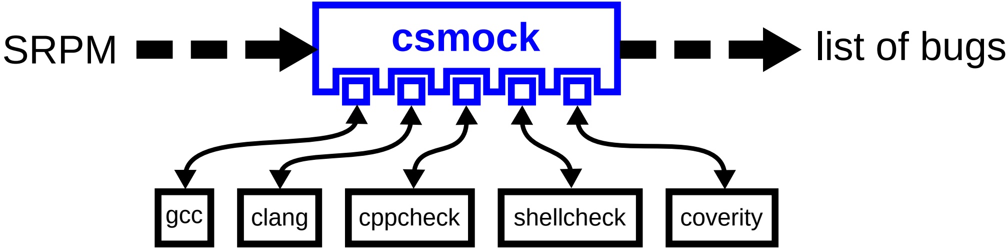
Static Analysis of RHEL in Numbers

- Analyzed 480 million LoC (Lines of Code) in 3700 packages.
- Preliminary scan of all RHEL 9 packages in February 2021.
- 98.6 % packages scanned successfully.
- Approx. 680 000 potential bugs detected in total.
- Approx. one potential bug per each 750 LoC.

Red Hat

Analysis of RPM Packages

- Command-line tool to run **static analyzers** on RPM packages.
- One interface, one **output format**, plug-in API for (static) analyzers.
- Fully open-source, available in Fedora and CentOS.



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 – Supported Static Analyzers

Tool	C	C++	C#	Java	Go	JavaScript	PHP	Python	Ruby	Shell
`gcc`	✓	✓								
`gcc -fanalyzer`	✓									
`clang --analyze`	✓	✓								
`cppcheck`	✓	✓								
`coverity`	✓	✓	✓	✓	✓	✓	✓	✓	✓	
`gitleaks`	✓	✓	✓	✓	✓	✓	✓	✓	✓	
`shellcheck`										
`snyk`			✓	✓	✓	✓	✓	✓	✓	
`unicontrol`	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`pylint`								✓		
`bandit`								✓		
`infer`	✓	✓								
`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 amount of incorrect results
- provide reproducible and consistent results
- be approximately as fast as ordinary compilation of the package
- support differential scans – detect added/fixed bugs in an update

Agenda

1 Linux Distribution, Reproducible Builds

2 Static Analysis of a Linux Distribution

3 **Dynamic Analysis of a Linux Distribution**

4 OpenScanHub

5 Differential ShellCheck

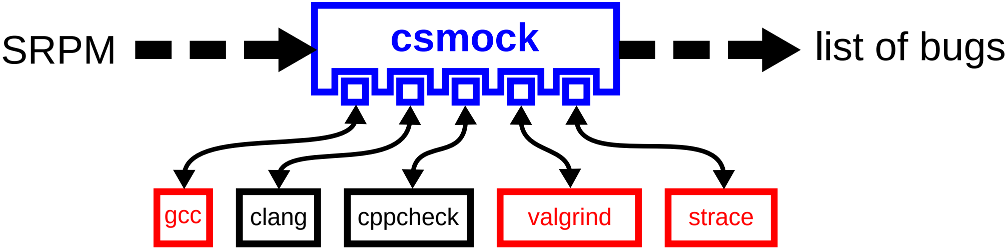
Dynamic Analysis

- Executes code in a modified run-time environment.
- Not so easy to automate as static analysis.
- Embedded in compilers: Address Sanitizer, Undefined Behaviour Sanitizer, . . .
- Standalone tools: Valgrind, strace, . . .
- Good to have some test-suite to begin with.

Red Hat

Dynamic Analysis of RPM Packages

- Requires an embedded test suite in the SRPM.
- `csmock` has experimental support for GCC sanitizers, Valgrind and strace:



Red Hat

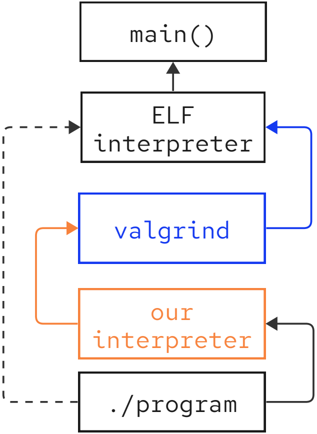
Dynamic Analysis of RPM Packages – Simple Approach

- Dynamic analyzers usually support tracing of child processes.
- Let's combine the tools together:
  - `valgrind --trace-children=yes rpmbuild --rebuild *.src.rpm`
  - `strace --follow-forks rpmbuild --rebuild *.src.rpm`
- But did we want to dynamically analyze `rpmbuild`, `bash`, `make`, etc.?
  - This makes the analysis extremely slow.
  - We get reports unrelated to `*.src.rpm`.

Red Hat

Dynamic Analysis of RPM Packages – Better Approach

- Build binaries that will launch the dynamic analyzer for themselves.
- Only binaries produced by `rpmbuild` will be executed through Valgrind.



Program Interpreter

- Program interpreter specified by shebang:

```
$ head -1 /usr/bin/dnf
#!/usr/bin/python3
```

```
$ /usr/bin/dnf [...]  ⟶  /usr/bin/python3 /usr/bin/dnf [...]
```

- Program interpreter specified by ELF header:

```
$ file /sbin/logrotate
/sbin/logrotate: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=...
```

- ELF interpreter can be set to a custom value when linking the binary:

```
$ file ./logrotate
./logrotate: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /usr/bin/csexec-loader, BuildID[sha1]=...
```

Wrapper of Dynamic Linker – Implementation

- We can use a compiler wrapper to instrument the build of an RPM package.
- csexec works as a wrapper of the system dynamic linker:
  https://github.com/csutils/cswrap/wiki/csexec
- $CSEXEC_WRAP_CMD can specify a dynamic analyzer to use.
- If the variable is unset, the binaries are executed natively.

```
$ export PATH="$(cswrap --print-path-to-wrap):$PATH"
$ export CSWRAP_ADD_CFLAGS=-Wl,--dynamic-linker,/usr/bin/csexec-loader
$ export CSEXEC_WRAP_CMD=valgrind
$ rpmbuild --rebuild *.src.rpm
```

- csexec runs the system dynamic linker explicitly (to eliminate self-loop):

```
./logrotate [...]  ⟶  valgrind /lib64/ld-linux-x86-64.so.2 ./logrotate [...]
```

Wrapper of Dynamic Linker – Evaluation

- Positives:
  - No completely unrelated bug reports.
  - Negligible impact on performance, excluding the time spent on analysis.
  - Minimal interference with commonly used testing frameworks.
  - Able to successfully run upstream test-suite of GNU Coreutils (without Valgrind).
- Negatives:
  - Some tests fail if we wrap them by Valgrind though:
    - a test that verifies the count of open file descriptors,
    - a test that intentionally sets non-existing `$TMPDIR`,
    - . . .

Agenda

1 Linux Distribution, Reproducible Builds

2 Static Analysis of a Linux Distribution

3 Dynamic Analysis of a Linux Distribution

4 **OpenScanHub**

5 Differential ShellCheck

Red Hat

OpenScanHub

- **OpenScanHub** is an open-source service for on-demand static and dynamic analysis.
- Uses `csmock` internally.
- Analysis of RPM packages and source code tarballs.
- Key Features
  - Support for differential scans.
  - Easily extensible through `csmock` plugins.
  - Reports from various analyzers are available in a single place.
- Available at `https://openscanhub.dev`.

Who should use it?

- Any developer can use it.
- It is used inside Red Hat to scan RHEL, OpenShift, OpenStack and other projects.
    - The goal is to scan all products shipped to our customers.
- We are currently in the process of building a public deployment of this service.

Agenda

1 Linux Distribution, Reproducible Builds

2 Static Analysis of a Linux Distribution

3 Dynamic Analysis of a Linux Distribution

4 OpenScanHub

5 **Differential ShellCheck**

Red Hat

Differential ShellCheck

- **Differential ShellCheck** performs differential analysis on shell scripts in your GitHub repository.
- Accessible as a GitHub Action.
  - Automatically checks for potential coding issues introduced by pull requests.
- Key features:
  - Auto-detection of shell scripts.
  - Statistics about fixed and added defects and their severity.
- Used by: `flatpak`, `systemd`, `strace`, `util-linux`, . . .
- Available at https://github.com/marketplace/actions/differential-shellcheck.

Red Hat

Q&A

Questions?