Static Analysis of a Linux Distribution

Kamil Dudka
Red Hat, Inc.
November 8th 2016
How to find programming mistakes efficiently?

0 users (preferably volunteers)

1 Automatic Bug Reporting Tool

2 code review, automated tests

3 static analysis!
Static Analysis

- is a good alternative to testing,
- can detect bugs fully automatically,
- can detect bugs before the code even runs!
Agenda

1. Terminology

2. Static Analysis of a Linux Distribution
Linux Distribution

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

- **upstream** SW projects – usually independent
- **downstream** distribution of upstream SW projects
  - Fedora and RHEL use 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.
Fedora vs. RHEL

- **Fedora**
  - new features available early
  - driven by the community (developers, users, ...)

- **RHEL** (Red Hat Enterprise Linux)
  - stability and security of running systems
  - driven by Red Hat (and its customers)
Where do RPM packages come from?

- Developers maintain source RPM packages (SRPMs).
- Binary RPMs can be built from SRPMs using `rpmbuild`:
  
  ```bash
  rpmbuild --rebuild git-2.6.3-1.fc24.src.rpm
  ```
- Binary RPMs can be then installed on the system:
  
  ```bash
  sudo dnf install git
  ```
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
  ```
Agenda

1 Terminology

2 Static Analysis of a Linux Distribution
Static Analysis of a Linux Distribution

- approx. 150 Million lines of C/C++ code in RHEL-7
- huge number of (potential?) defects in certain projects
- thousands of packages developed independently of each other
- no control over technologies and programming languages
- no control over upstream coding style
Which static analyzers?

- Not many of them are ready for scanning a Linux distribution.

- Some analyzers are tweaked for a particular project (e.g. sparse for kernel).

- How to use multiple static analyzers easily?

  - The `csmock` tool provides a common interface to GCC, Clang, Cppcheck, Shellcheck, Pylint, and Coverity.

  - Besides C/C++, Java, and C#, **Coverity** now also analyzes dynamic languages (JavaScript, PHP, Python, Ruby).
Example – Defects Found by Coverity Analysis

Error: **IDENTIFIER_TYPO**: [#def1]
anaconda-21.48.22.90/pyanaconda/ui/gui/spokes/source.py:1388: **identifier typo**: Using "mirorlist" appears to be a typo:
* Identifier "mirorlist" is only known to be referenced here, or in copies of this code.
* Identifier "mirrorlist" is referenced elsewhere at least 27 times.
anaconda-21.48.22.90/pyanaconda/packaging/__init__.py:1046: **identifier_use**: Example 1: Using identifier "mirrorlist".
anaconda-21.48.22.90/pyanaconda/packaging/yumpayload.py:726: **identifier_use**: Example 4: Using identifier "mirrorlist".
anaconda-21.48.22.90/pyanaconda/packaging/yumpayload.py:335: **identifier_use**: Example 5: Using identifier "mirrorlist".
anaconda-21.48.22.90/pyanaconda/ui/gui/spokes/source.py:1388: **remediation**: Should identifier "mirorlist" be replaced by "mirrorlist"?

```
# 1386|           url = self._repoUrlEntry.get_text().strip()
# 1387|           if self._repoMirrorlistCheckbox.get_active():
# 1388|->             repo.mirorlist = proto + url
# 1389|           else:
# 1390|             repo.baseurl = proto + url
```

Error: **NESTING_INDENT_MISMATCH**: [#def2]
infinipath-psm-3.3-19_g67c0807_open/psm_diags.c:284: **parent**: This 'if' statement is the parent, indented to column 5.
infinipath-psm-3.3-19_g67c0807_open/psm_diags.c:285: **nephew**: This 'if' statement is nested within its parent, indented to column 7.
infinipath-psm-3.3-19_g67c0807_open/psm_diags.c:286: **uncle**: This 'if' statement is indented to column 7, as if it were nested within the preceding parent statement, but it is not.

```
# 284|       if (src == NULL || dst == NULL)
# 285|         if (src) psmi_free(src);
# 286|->       if (dst) psmi_free(dst);
# 287|       return -1;
# 288|   }
```

Error: **COPY_PASTE_ERROR** (CWE-398): [#def3]
gnome-shell-3.14.4/js/ui/boxpointer.js:517: **original**: "resX -= x2 - arrowOrigin" looks like the original copy.
gnome-shell-3.14.4/js/ui/boxpointer.js:536: **remediation**: Should it say "resY" instead?

```
# 534|   } else if (arrowOrigin >= (y2 - (borderRadius + halfBase))) {
# 535|     if (arrowOrigin < y2)
# 536|->       resX -= (y2 - arrowOrigin);
# 537|     arrowOrigin = y2;
# 538|   }
```
Example – A Defect Found by ShellCheck

Error: SHELLCHECK_WARNING: [#def4]
/etc/rc.d/init.d/squid:136:10: warning: Use "${var:?}" to ensure this never expands to /* . [SC2115]
# 134| RETVAL=$?
# 135| if [ $RETVAL -eq 0 ] ; then
# 136|->               rm -rf $SQUID_PIDFILE_DIR/*
# 137|                 start
# 138|         else

https://github.com/koalaman/shellcheck/wiki/SC2115
What is important for developers?

The static analysis tools need to:

- be fully automatic
- provide reasonable signal to noise ratio
- be approximately as fast as compilation of the package
- deliver results in a predictable amount of time ⇒ timeouts!
Research Prototypes

- Researchers are done when their tool works on a few examples of their choice. (phase 0)

- SW companies are interested in tools that can reliably process a significant amount of their code base. (phase 1)

- 99% of work on developing a successful tool is the transition: phase 0 $\rightarrow$ phase 1

- Competition on Software Verification (SV-COMP):
  
  https://sv-comp.sosy-lab.org/2016/results/results-verified/
Priority Assessment Problem

- Developers say:
  
  "I have 200+ already known bugs in my project waiting for a fix. Why should I care about additional bugs that users are not aware of yet?"

- Not all defects are equally important to be fixed!

- Scoring systems like CWE (Common Weakness Enumeration)

- ... but none of them is universally applicable.
Differential scans

- We know that our packages contain a lot of potential bugs.
- It is easy to create new bugs while trying to fix existing bugs.
- Which bugs were added/fixed in an update of something?
- An example using the csbuild utility – demo (GNU nano):

```
  csbuild -c "make -j9"
  csbuild -g v2.7.0..master -c "make -j9"
  csbuild -g v2.7.0..master --git-bisect \ 
      -c "make clean && make -j9"
```
Upstream vs. Enterprise

Different approaches to (differential) static analysis:

**Upstream** – Fix as many defects as possible.
- False positive ratio increases over time!

**Enterprise** – Verify code changes in ancient SW.
- 5–10% of defects are usually detected as new in an update.
- 5–10% of them are usually confirmed as real by developers.
Processing the Results of Static Analysis

- Some tools come with a user interface for waiving defects.
- Per-defect waivers do not scale for a Linux distribution.
- Certain developers prefer to use terminal over web browser.
- Utilities processing text line-by-line are not optimal for this:
  
  \[
  \text{grep} \rightarrow \text{csgrep} \\
  \text{sort} \rightarrow \text{cssort} \\
  \ldots
  \]

  https://github.com/kdudka/csdiff
Continuous Integration

- It is expensive to fix bugs detected late in the release schedule.
- It is difficult and risky to fix bugs in already released products.
- We would like to catch bugs at the time they are created.
- An example using the csbuild utility – demo:

```
csbuild -c "./buildconf && ./configure && make -j9" \ 
   --install libtool --git-bisect \ 
   --gen-travis-yml > .travis.yml

git add .travis.yml
git commit -m "notify me about newly introduced defects"
git push
```
Slides Available Online