redhat

Static Analysis of a Linux Distribution

Kamil Dudka <kdudka@redhat.com>
Red Hat, Inc.

November 26th 2015

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

redhat

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`:

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

- Binary RPMs can be then installed on the system:

```
sudo dnf install git
```

redhat

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

redhat

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

redhat

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).
- Using a single static analyzer appeared to be insufficient.
- How to combine multiple static analyzers efficiently?
- Currently supported by csmock:
  GCC, Clang, Cppcheck, Shellcheck, Pylint, Coverity

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 $\Longrightarrow$ timeouts!

redhat.

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 $\longrightarrow$ phase 1
- example – Predator:
  http://www.fit.vutbr.cz/research/groups/verifit/tools/predator

redhat.

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.

redhat

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:

```
csbuild -c "make -j5"
csbuild -g curl-7_40_0..master -c "make -j5"
csbuild -g curl-7_40_0..master --git-bisect \
        -c "make clean && make -j5"
```

redhat

Upstream vs. Enterprise

Different approaches to (differential) static analysis:

Upstream

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

Enterprise

- Need to 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:

`grep` ⟶ `csgrep`
`sort` ⟶ `cssort`
...

https://github.com/kdudka/csdiff

redhat.

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 -j5" \
        --install libtool --git-bisect \
        --gen-travis-yml > .travis.yml

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