Static Analysis of a Linux Distribution

Kamil Dudka
Red Hat, Inc.
November 6th 2017
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!
Agenda

1. Code Review
2. Dynamic Analysis
3. Static Analysis
4. Linux Distribution
5. Static Analysis of a Linux Distribution
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
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, ...)

- fuzzing (feeding programs with unusual input), e.g. oss-fuzz
Static Analysis

- does not need to run the code
- does not need any test-suite
- can detect bugs fully automatically
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
Agenda

1. Code Review
2. Dynamic Analysis
3. Static Analysis
4. Linux Distribution
5. 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`:
  
rpmbuild --rebuild git-2.6.3-1.fc24.src.rpm

- Binary RPMs can be then installed on the system:
  
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. Code Review
2. Dynamic Analysis
3. Static Analysis
4. Linux Distribution
5. 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 programming languages and coding style used by upstream
Which static analyzers?

- Some analyzers are tweaked for a particular project (e.g. sparse for kernel).
- Relying on a single static analyzer is insufficient.
- 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: **NESTING_INDENT_MISMATCH**: [#def1]
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.

```c
#define DEF1

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): [#def2]
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?

```javascript
#define DEF2

534|               } else if (arrowOrigin >= (y2 - (borderRadius + halfBase))) {
535|                 if (arrowOrigin < y2)
536|->               resX -= (y2 - arrowOrigin);
537|               arrowOrigin = y2;
538| }
```

Error: **IDENTIFIER_TYPO**: [#def3]
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 "mirorlist" 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"?

```python
#define DEF3

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
```
What is important for developers?

The static analysis tools need to:

- be fully automatic
- provide reasonable signal to noise ratio
- results need to be reproducible and consistent
- be approximately as fast as compilation of the package
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 bugs 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?
Example – Differential Scan of logrotate (1/2)

- On September 19 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'

- On September 20 we agreed on a fix and pushed it (https://github.com/logrotate/logrotate/pull/149):

- Release of logrotate-3.13.0 scheduled on October 13th...
Example – Differential Scan of logrotate (2/2)

- On October 12th (a day before the release) I ran a differential scan with the csbuild utility – demo:

```bash
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):

```bash
csbuild -c $BUILD_CMD -g origin..master --print-fixed
```
Upstream vs. Enterprise

Different approaches to static analysis:

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

**Enterprise** – Verify code changes in ancient SW.
- 5–10% of bugs are usually detected as new in an update.
- 5–10% of them are usually confirmed as real by developers.
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:

```bash
csbuild --install 'automake libpopt-devel'
   --prep-cmd 'autoreconf -fiv && ./configure'
   --build-cmd 'make clean && make -j9'
   --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

https://kdudka.fedorapeople.org/muni17.pdf