ONAP static code analysis by Coverity Scan

Introduction & Setup

ONAP Joint Subcommittee Meeting · Antwerp, Belgium · September 26–27, 2019
Artem Naluzhnyy <A.Naluzhnyy@samsung.com> · Samsung R&D Institute Poland
Coverity Scan – static code analysis SaaS

- Free for Open Source projects; used by:
  - Linux kernel
  - LibreOffice
  - FreeBSD OS

- Low false positive ratio

- Shows events chain contributing to a defect

- Does not require source/build code changes to run

- Supported languages:
  - Java
  - JavaScript / TrueScript
  - Python
  - Scala
  - C / C++
  - and more…
Critical checkers

- API usage errors
- Best practice coding errors
- Buffer overflows
- Build system issues
- Class hierarchy inconsistencies
- Code maintainability issues
- Concurrent data access violations
- Control flow issues
- Cross-site request forgery (CSRF)
- Cross-site scripting (XSS)
- Deadlocks
- Error handling issues
- Hard-coded credentials
- Incorrect expression
- Insecure data handling
- Integer handling issues
- Integer overflows
- Memory corruptions
- Illegal memory accesses
- Null pointer dereferences
- Path manipulation
- Performance inefficiencies
- Program hangs
- Race conditions
- Resource leaks
- Rule violations
- Security best practices violations
- Security misconfigurations
- SQL injection
- Uninitialized members

See also Coverity checkers [registration required]
**Concepts**

- **Coverity project** ⇔ ONAP repository
  - Naming conventions example for "sdc/dcae-d/fe" ONAP repo
    → "onap-sdc-dcae-d-fe" Coverity project
    → "sdc-dcae-d-fe-coverity" ONAP Jenkins job

- **Coverity project component** ⇔ Subset of ONAP repo sources (e.g. "BE" or "FE")
  - Developers may subscribe to specific component defects only

- **Coverity Scan user roles**
  - "Contributor/Member" → ONAP developers (review&comment defects)
  - "Maintainer/Owner" → ONAP PTL / admin (configure components, grant permissions to developers)

- **Coverity Scan quotas** (1M+ LOC → 1 build per day)

- See also Coverity [glossary] [registration required]
Architecture overview

1. Get ONAP component sources
2. Download Coverity Scan toolset
3. Analyse "mvn install" by Coverity toolset
4. Submit analysis binary output
5. Analysis completed notification email
6. Top 20 new defects summary email
7. Access to all defects

Coverity Scan service
Setup workflow overview
Setup workflow: Coverity project registration

Project Name: onap-so-liba
Role: Maintainer/Owner
Language: Java
Repository URL: https://gerrit.onap.org/r/so/liba.git
License: Apache - Apache License
Project Access: Project summary and defects are viewable in read-only mode by all users
Homepage URL: https://onap.org/
Reference URL: showing your association with the project - example - https://github.com/daksh/vyasa/MyHelloWorld/commit/master
Additional information: so/liba is a component of Open Networking Automation Platform - an open source networking project hosted by the Linux Foundation.

Project token: XXXjV5T-oe0KSeaF7r
Setup workflow: ONAP Jenkins configuration

- project:
  name: 'portal-coverity'
  jobs:
  - 'onap-gerrit-maven-coverity'
    cron: '@daily'
    max-git-repo-age-hours: 48
    build-node: 'ubuntu1604-builder-4c-4g'
    project: 'portal'
    project-name: 'portal'
    branch: 'master'
  mvn-settings: 'portal-settings'
  mvn-params: '-Dmaven.test.skip=true'
  # Additional mvn-params: '-DskipTests'
  coverity-project-name: 'onap-portal'
  coverity-token: 'SrGGJp9T1nn2sF72XQ'
  coverity-user-email: 'my.email@example.com'
  coverity-search-paths: >
  ecomp-portal-FE-os/client
  ecomp-portal-FE-os/mock
  ecomp-portal-FE-os/server
  ecomp-portal-widget-ms
  ecomp-portal-FE-common/client/app
  coverity-search-exclude-regexes: >
  /node_modules/
  /bower_components/
  /bower_components_external/
  dry-run: false

Jenkins > portal-coverity > #32 > Parameters

Enable Auto Refresh

Dry Run

Do not submit results to Coverity Scan server at the end of the build.
**Setup workflow: Define project components**

Defining components is a great way to focus your defect fixing efforts. Once defined, Covery Scan groups defects under their respective components. Components can also be used to have the analysis ignore certain parts of the code base such as third-party code.

The path adheres to regular expression syntax. For example, to exclude defects from `/usr/lib`, `/any-source/external/lib/` or test code, add `/usr/lib/.*`, `./external/lib/.*`, or `./test/.*` respectively.

<table>
<thead>
<tr>
<th>Component name</th>
<th>Pattern</th>
<th>Ignore in analysis</th>
<th>Remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>tests</td>
<td><code>.*src/test\.*</code></td>
<td>Yes</td>
<td>Remove</td>
</tr>
<tr>
<td>3rd-party</td>
<td><code>.*node_modules\.*</code></td>
<td>Yes</td>
<td>Remove</td>
</tr>
<tr>
<td>system</td>
<td><code>./usr\.*</code></td>
<td>Yes</td>
<td>Remove</td>
</tr>
<tr>
<td>ecomp-portal-BE-common</td>
<td><code>./ecomp-portal-BE-common\.*</code></td>
<td>No</td>
<td>Remove</td>
</tr>
<tr>
<td>ecomp-portal-BE-os</td>
<td><code>./ecomp-portal-BE-os\.*</code></td>
<td>No</td>
<td>Remove</td>
</tr>
<tr>
<td>ecomp-portal-FE-common</td>
<td><code>./ecomp-portal-FE-common\.*</code></td>
<td>No</td>
<td>Remove</td>
</tr>
</tbody>
</table>
Setup workflow: Grant permissions to developers

Invite others to join the project:
Enter email addresses to invite others to join your project. Invites can also view source defects.

OR

Coverity Scan: onap-portal

Project Name: onap-portal
Lines of code analyzed: 115,077
On Coveryity Scan since: Jul 01, 2019
Last build analyzed: 2 days ago

Language: Java
Secondary Language: Java
Repository URL: https://git.onap.org/portal/
Homepage URL: https://onap.org/
License: Apache (Apache License)

Want to view defects or help fix defects?

Add me to project

Analysis Metrics
Version: e10ac25

Sep 10, 2019 115,077 Lines of Code Analyzed
Last Analyzed
2.83 Defect Density
Setup workflow: Troubleshooting

• Jenkins build logs:

  • "cov-int/coverity-scan-analysed-files.txt"
    → files sent for analysis

  • "cov-int/scm-untracked-files.txt"
    → 3rd-party and auto-generated sources

  • "cov-int/failed_jsp/*"
    → errors in .jsp files

  • "cov-int/build-log.txt"
    → Coverity toolset build log
Interface for developers

1. taint_path.param: Parameter org.userId receives the tainted data.

   ```java
   public List<FunctionalMenuItem> getFunctionalMenuItemsForUser(String orgUserId) {
       // m represents the functional menu items that are the leaf nodes
       // ml represents the functional menu items for all the nodes

       // Divide this into 2 queries: one which returns the bottom-level menu items
       // associated with Restricted apps.
       // and one that returns all the other menu items. Then we can easily add the
       // boolean flag
       // restrictedApp to each FunctionalMenuItem, to be used by the front end.
       String sql = "SELECT DISTINCT ml.menu_id, ml.column_num, ml.text, ml.parent_menu_id, ml.url, m.active_ym 
                   WITH fn_menu_functional m, fn_menu_functional ml, fn_menu_functional_ancestors a, 
                   fn_menu_functional_roles mr, fn_user u, fn_user_role ur " + "WHERE " + "u.org_user_id =" + orgUserId;

   sql taint: Insecure concatenation of a SQL statement. The value org.userId is tainted.

   Perform the following to guard against SQL injection attacks:
   - Parameterize the SQL statement.
   - Bind the tainted value to the parameter.
   ```

   More Information
Coverity Scan service issues

• Language support:
  • Go is coming
  • Kotlin is on the roadmap
  • No support of Clojure, Erlang and Lua

• Missing source code branches support

• The service is under maintenance at the moment (2019–09–19):
  • WEB/API may be unavailable or read-only
  • specific features may be disabled
Implementation status

Done

• "onap-gerrit-maven-coverity"
  JJB job template [CIMAN-260]

• Analysed ONAP components

• Active Jenkins jobs

• Wiki page

To Do

• Cover more ONAP components

• Guide/demo for developers

• Optimize JJB template:
  • bandwidth & build time
  • secure Coverity tokens
Open questions

- Who should manage Coverity submission errors?
- How to manage Coverity Scan project tokens in our Jenkins?
- Should we launch tests by default?
- Should we analyse test sources by default?
- Should we analyse 3rd-party sources by default?
- Should we analyse auto-generated sources by default?
The presentation video is available here.

Need help?

• Check ONAP Wiki
• Assign Jira ticket to @Naluzhnyy
• Ask Coverity Scan community
• Contact <scan-admin@coverity.com>