Axel Naumann

Coverity and ROOT

Outline

- Static Code Analysis
- Coverity
- Reporting Tools, Report Quality
- "Demo": Examples

Code Analysis

Relevance

- US Dept Homeland Security contract for 150 Open Source tools
- Consequence: 6000 bugs fixed
  - "security hole in X Windows that allows any user with a login to gain root privileges"
  - CNET: "Key bugs in core Linux code squashed"
  - Samba: "errors found by Scan can save the reputation of a project"

Static Code Analysis

- Only looking at sources
- Code does not run
- "Replacement" for compiler
- Reflects theoretically possible code path

```
if (a) delete p;
...
if (x) p->call();
```

Static vs Dynamic Analysis

- Dynamic closer to reality:
  all *actual* code paths
- Static more thorough:
  all *possible* code paths
- Major issues with static analysis:
  - false positives ("noise")
  - cannot distinguish relevant from hypothetical
  - complex, time consuming analysis

The Company Behind the Tool

Covertity

Coverity: Open Source Scans

- Famous for their annual open source reports with static checker *Prevent*, e.g.
  - Linux, FreeBSD, NetBSD
  - Apache, Samba, Squid, Postfix
  - MySQL, PostgreSQL
  - Perl, Python, PHP, gcc
  - OpenSSH, libjpeg, libtiff, pcre
  - Firefox, Thunderbird

Coverity: Commercial Clients

- >900 customers
- ARM, Philips, RIM, Samsung, UBS, Symbian, Palm, RSA, Yahoo, McAfee
- Used to large (i.e. huge) software bases, n*10M LOC
- Used to funny build systems

Coverity as Company

- Founded 2002
- About 150 employees
- Headquarters in San Francisco
- European sales office (UK)
- Engineers currently based in US

First Contact

- I want code quality for ROOT, CERN
- They want increased visibility in Europe
  - "Free check for you, press release for us"
- Need more than Coverity's free open source support
  - Responsiveness
  - Customization
  - "We want it, too": cernvm, ALICE, CMS,…

Offer

Waiting for written offer from Coverity; agreement:

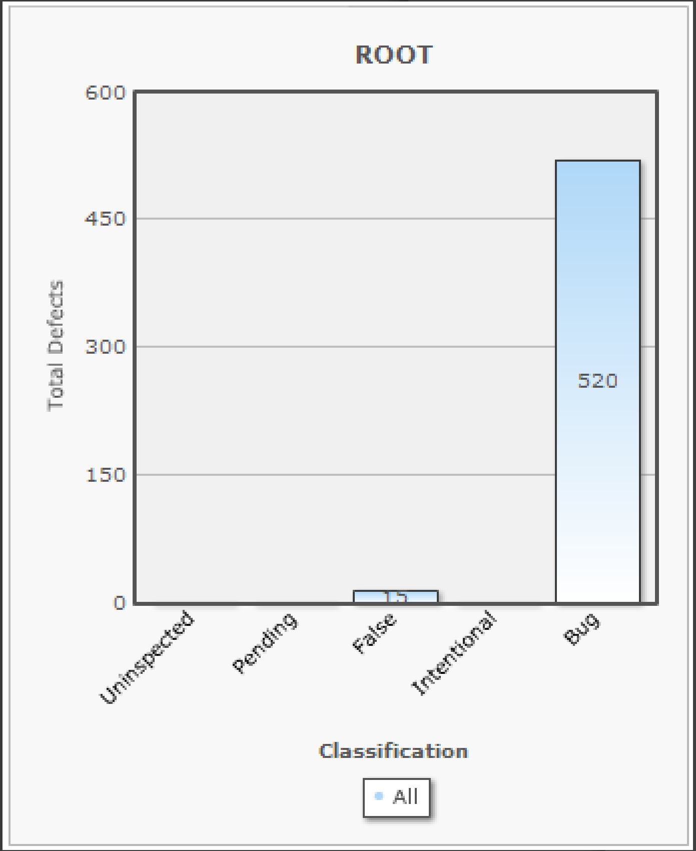
- Update code and re-run checker on their servers "for life", for free
- Training: one engineer for one week
  - upload code
  - configure build
  - run checker

Visualization and Tracking Of Reports

Reports

Reporting Tools

- Web interface
- Code hierarchy
  - products ("ROOT")
  - components ("Math", "CINT")
  - files, functions, individual
- Evolution over time
- Used to assign, mark, comment defects
- Customizable DB queries
- Graphs



Annotated Source

- Reports embedded in actual source so you see what’s wrong (see demo)
- Identifiers linked, to jump into call etc
- enormous amount of data: each defect has its own source "view"

Triage

- Flag defects as
  pending / false / intentional / bug
- Determine action
  fix / resolved / ignore
- Assess severity
- Assign to owner

Report Quality

- Thousands of reports for ROOT's 2MLOC
- Of the handled defects, 35 times more true than false reports
- Reason e.g.: scanner looks for context

```
case kFunc:
   func();
   // intentional fallthrough
case kWhatever: ...
```

- Watch out: defects are painful!

Checkers

- Checker analyze code for defect type
  - uninitialized variables
  - use after delete
  - unused code
  - array bounds
  - stack size
  - …
- Documentation with each checker

Checkers' Reach

- C / C++ checkers

  bad free, infinite loop, return local, pass by value, missing break…
- Security

  overflows (string, int), unchecked user input, time of check vs. time of use…
- Concurrency

  atomicity, lock, sleep,…

Examples

- Historical Samba reports
- Examples from ROOT