Lessons Learned in Software Assurance Evaluations

W. Konrad Vesey
National Security Agency
Center for Assured Software
SSTC
1 May 2008

Center for Assured Software

- The mission of the Center for Assured Software is to define and promulgate guidance on software development, evaluation, and acquisition practices that will increase the assurance of DoD software.

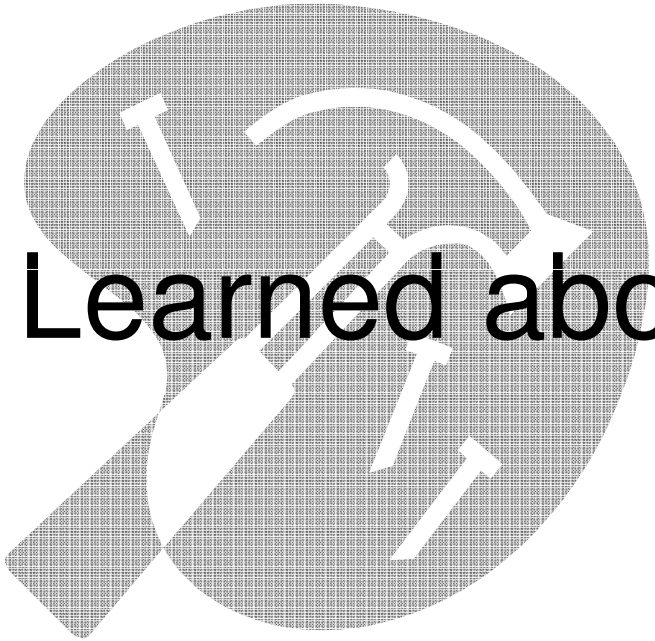
Software Assurance Evaluations

- A software assurance evaluation is a determination of the degree of confidence that software performs as intended, performs no unauthorized functions, and contains no exploitable weaknesses.

Lessons Learned

- The CAS has been developing evaluation techniques to define and promote a highly automated, reproducible, well-documented methodology that can serve as the basis for a Joint approach to software security measurement.
- We keep our hands dirty looking at real code
- We make a lot of mistakes

Lessons Learned about Tools



Lessons Learned about Tools

- Tools don't find the same things
  - Even when they claim to
  - Each tool does a few things "well"
- Tools that find the same things don't have the same strengths
  - Even when they claim to
- Tools that find the same things don't call them the same things
  - "Even when they claim to" (in anticipation of CWE ambiguities)
- Tools that find the same things and call them the same things don't report them the same way
  - Source location vs. sink location
  - All paths are one finding, each path is one finding
- Tools vary in their approach to reporting
  - Include low-confidence hits = results approximate a lower bound on correctness
  - Trim low-confidence hits = results approximate an upper bound on correctness

Lessons Learned about Tools

- Tools have different out of the box configurations
  - Everything on vs. everything off
  - Know your tools inside and out
- Tools may not be able to handle large codebases
  - Break up large codebases into smaller projects before running static analysis tools
- Integrating tools into build scripts can be non-trivial
  - Tools don’t support arbitrary build processes
- Know what your tool did
  - Make sure it didn’t skip large sections of the code

Lessons Learned about Tools

- Some code is so bad, tool based evidence is useless
  - If the tool finds few flaws: the tool's ability to analyze the code is suspect, it probably got confused
  - If the tool finds many flaws: the tool's ability to analyze the code is suspect, it probably got confused
  - Note: "bad" code ≠ "flawed" code
- Tool-based evidence is meaningless outside of context
  - Is the evidence presented relevant to the claim being made?
  - What is the claim being made?
  - How strong is the tool in the areas evaluated?
  - What was the tool's configuration when it was run?
  - Did the tool look at all of the code?
  - Does the tool report its findings in ways that readily map to the claim?
  - Does the evidence presented call out the relevant findings?
  - "Raw" tool output is impossible for a certification review board to interpret outside of the context of all of the above

Lessons Learned about Teams

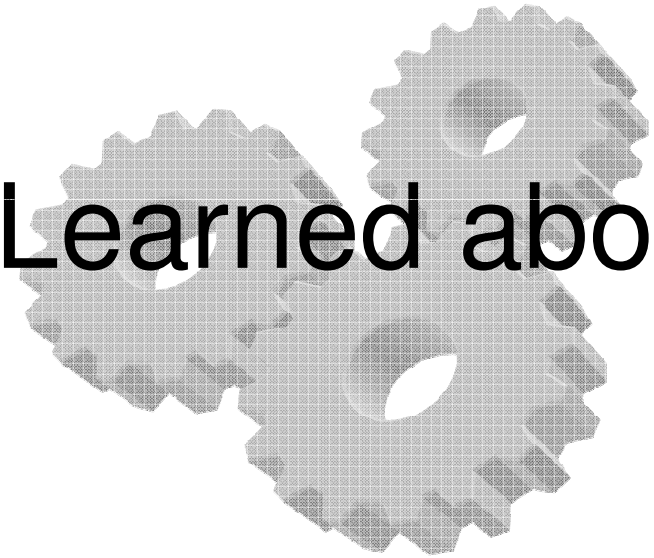
Lessons Learned about Teams

- Different evaluators have different levels of skill, experience, and competency
  - Coding experience doesn't mean security knowledge
  - Don't assume everyone is keeping track of their time in the same way – review early
  - Capture rules of thumb to aid the less experienced and keep the team consistent
- It's more efficient for an evaluator to review a diverse set of findings in the same file than to review the same type of finding over a set of files
- Manage the team as a team
  - Don't treat them as a collection of gurus (even if they are)
  - Frequent oversight for consistency
  - Share interesting or difficult findings
  - Spend the effort to get and keep everyone on the same page

Lessons Learned about Teams

- Keep the evaluation team comfortable
  - Evaluations can be tedious, add variety to individual tasking over the course of the evaluation
  - Meetings should have cookies and an agenda
    - If the agenda is weak, substitute donuts (for the agenda, not for the cookies)
- Compiling a single report from the contributions of multiple team members is an editorial task for which few security analysts are trained
  - Established document conventions and a style guide grow in importance the more contributors there are
  - Keep report drafts under version control so recovery/rollback is possible
  - Use each report to refine boilerplate sections that can be dropped in without detailed editing
  - Limit the number of authors to one or two (but start them earlier)
- The state of tools today is such that analysts do not have to be security experts in order to be effective

Lessons Learned about Process



Lessons Learned about Process

- Good understanding tools are invaluable
  - Program understanding, source code indexing/browsing, and visualization tools help analysts verify tool findings faster and more accurately
  - Understanding the significance of different parts of the code can help determine the impact/risk of individual findings
- Use collaboration and workflow tools to keep the team communicating and recording key metrics
- Infrastructure is vital
  - Having the right processing power and network bandwidth is critical for large codebases
  - Current tools consume a lot of resources
- Establish delivery requirements up front
  - If you don't you'll get the code delivered on a headless workstation with a nonfunctioning disk drive with no login password
  - If you don't you'll get the code delivered on a tape media format you haven't seen in decades

Lessons Learned about Process

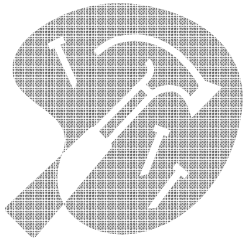
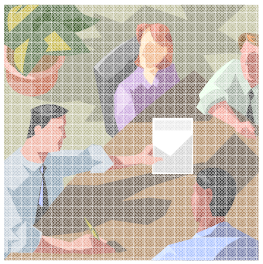
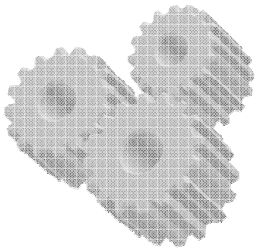
- Poor configuration management makes analysis difficult to impossible
  - Half-implemented features, files with identical names but different functional content, different versions of the source mixed up in the same folders
- Software projects are not homogeneous – integrated tool support is rare
  - Mixture of third-party binaries, open source, custom code, different languages, script and compiled code mean your tools may only be able to analyze a portion of the software
  - Lack of access to third-party source code can be a barrier to analysis
- If you can't build it, you (probably) can't analyze it
  - Plan project milestones from the time you have verified the code is buildable
- Don't introduce a new tool in an actual evaluation
  - Don't use an actual evaluation as an opportunity to debug your internal tools either

Lessons Learned about Process

- Understand the build process
  - Permits breaking up a large codebase into subprojects that tools can handle
  - Know what was included in the build: avoid reviewing code that isn't part of the product being evaluated
  - Unfortunately not aware of adequate tool support for this
- Standardize evaluation environments
  - Virtualization tools are useful for this
- Keep a journal
  - Capture evaluation environment setup and tools configuration data as you go
- Don't rush to analysis
  - Spend the necessary time to make sure the tools ran as expected
- Capture your lessons learned for continuous process improvement

Lessons Learned

- It is possible to evaluate a code base of significant size in an cost-efficient and timely manner

Lessons Learned in Software Assurance Evaluations

W. Konrad Vesey
National Security Agency
Center for Assured Software
SSTC
1 May 2008