Static Analysis as Part of the Code Review Process

Thomas L. "Trey" Jones, CISSP, CEH

Toll Road OCR Exploit

Introduction

- Requires organization wide planning
  - a plan for who will conduct security reviews
  - when the reviews will take place
  - how to act on the results

- Static code analysis (SCA), both manual and automated (tools) should be part of the plan

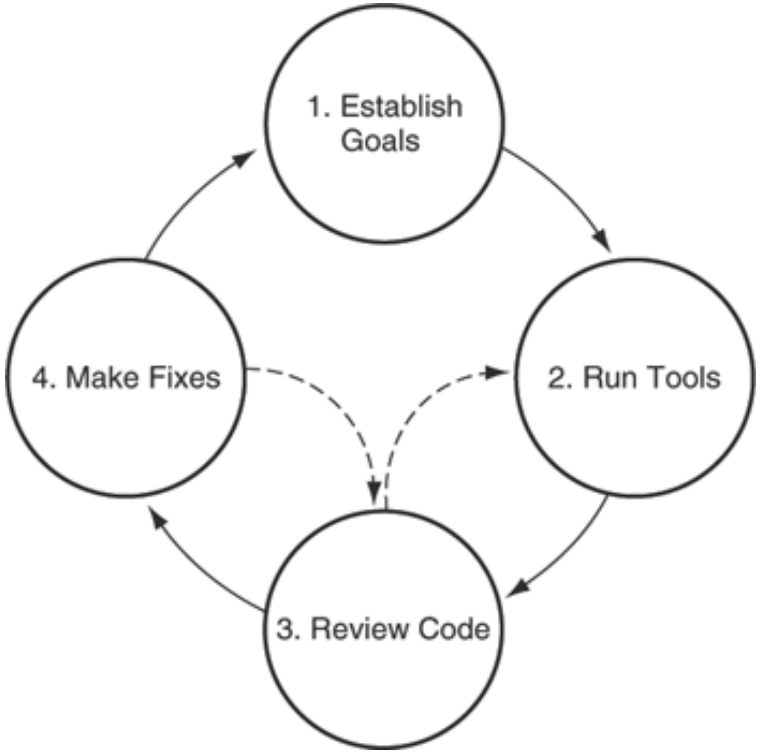
Performing a Code Review

- Driven by:
  - Need to find a few exploitable vulnerabilities to prove that additional security investment is justified.
  - The team eventually has to make an initial pass through the code to do a security retrofit (not considered at beginning of project).
  - At least once in every release period, every project should receive a security review to account for new features and ongoing maintenance work.

- Of the three, the second requires by far the largest amount of time and energy.

The Review Cycle

- The four major phases in the cycle are:
  - Establish goals
  - Run the static analysis tool
  - Review code (using output from the tool)
  - Make fixes



Establish Goals

- A well-defined set of security goals will:
  - Help prioritize the code that should be reviewed
  - Criteria that should be used to review it.
- Goals should come from an assessment of the software risks
  - Need enough high-level guidance to prioritize potential code review targets.
  - Set review priorities down to the level of individual programs, don't subdivide any further
  - Run static analysis on at least a whole program at a time
- Make sure reviewers understand the purpose and function of the code being reviewed.
  - A high-level description of the design is a must.
  - Review the risk analysis results relevant to the code.
- Without the above an ad-hoc result

"Uh, err, the OWASP Top Ten?" Bad answer.

Run Static Analysis Tools

- Gather the target code and build files
- Ensure all code is syntactically correct
  - Compiles
  - Links (this insures you know where libraries are located)
- Configure the tool to report the kinds of problems that pose the greatest risks, and
- Disable checks that aren't relevant.

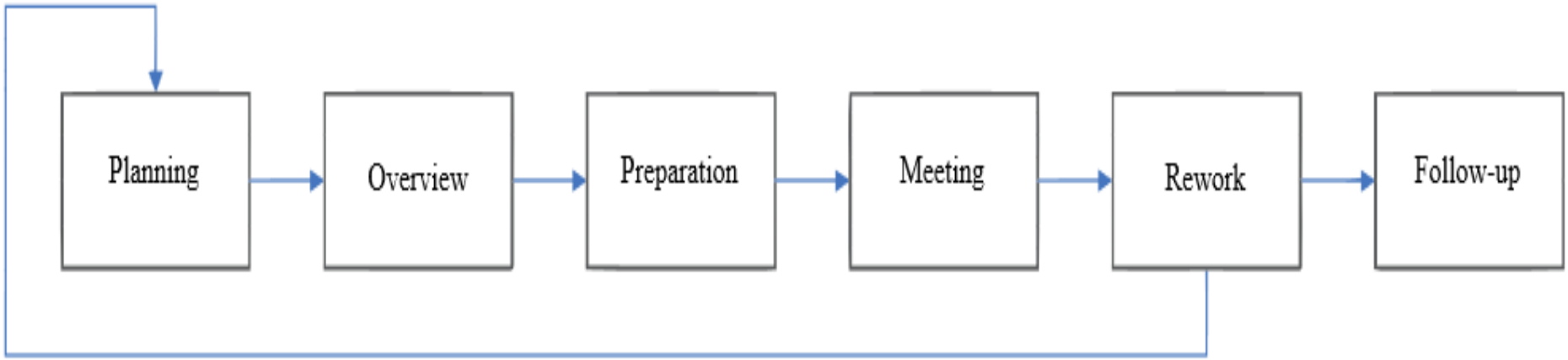
Run Static Analysis Tools$_2$

- Add custom rules to detect errors that are specific to the program being analyzed.
  - Include a tool that checks for compliance to organizational coding standards
- If your organization has a set of secure coding guidelines, encode as custom rules.
  - A static analysis tool won't, by default, know what constitutes a security violation in the context of your code.
- The output from this phase: a set of raw results for use during code review.

Review Code (aka Peer Review)

- Team puts "eyes on code"
  - Peer review and static analysis are complimentary techniques.
- Fagan Inspection
  - "A defect is an instance in which a requirement is not satisfied."
- Typical code review rates: 125-150 lines of code per hour
  - for critical software (such as safety critical embedded software) may be too fast to find errors.
  - Industry data indicates that code reviews can accomplish at most an 85% defect removal rate with an average rate of about 65%.
- Empirical studies provided evidence that up to 75% of code review defects affect software evolvability rather than functionality
  - Code reviews an excellent tool for software companies with long product or system life cycles.

Fagan Inspection



Review Code$_2$

- Don't just look at tool results
  - Use tool results to assess problems
  - If false positive, document why
- The team will likely spot other False Negatives
  - Consider adding new SCA rules for future use.
- Questions to consider
  - Do mitigating factors prevent the code from being vulnerable?
  - Is the source of untrusted data actually untrusted?
  - Is the scenario hypothesized by the tool actually feasible?
- Collaborate with the author or owner of the code when context is needed

Make Fixes

- Programmers respond to the feedback from a security review:
  - Does security matter to them?
    - Is security software a prerequisite for releasing their code? Anything less competes with adding new functionality, fixing bugs, and making the release date.
  - Do they understand the feedback?
- Addressing the results of security reviews must be in the Software Development Lifecycle.

Process must verify “fixes”!

Steer Clear of the Exploitability Trap

- Three possible verdicts that a piece of code might receive during a security review:
  - Obviously exploitable
  - Ambiguous
  - Obviously secure
- Is ambiguity due to poor coding styles?
- Ambiguous is NOT a reason to waste time debating
  - Take more time than fixing the problem
  - Building exploits is a specialized skill
- Five Lame Excuses for Not Fixing Bad Code

If a piece of code isn't obviously secure, make it obviously secure

Adding Security Review to an Existing Development Process

- The famous 2002 memo from Bill Gates titled "Trustworthy Computing" is a perfect example of management direction. In the memo, Gates wrote:
  - So now, when we face a choice between adding features and resolving security issues, we need to choose security.

- Who runs the tool?
- When is the tool run?
- What happens to the results?

Who runs the tool and when?

- Developers
  - Regularly as they build and compile code
- Security Review Team
  - Planned by schedule (major milestones)
  - Don't do one major tool run / code review at product finalization.

Passing Security must be part of Acceptance Criteria!

Static Analysis Metrics$_1$

- Metrics derived from static analysis results, if done right, help quantify the amount of risk associated with a piece of code.
  - By manually auditing enough results, a security team can predict the rate at which false positives/negatives occur and extrapolate number of true positives from raw results.
- Problem: There is no good way to sum up the risk posed by a set of vulnerabilities.
  - Proposal: Use static analysis output to focus security efforts and as an indirect measure of the process used to create the code.

Static Analysis Metrics$_2$

- Vulnerability density is a horrible measure on its own.
  - The way a program is written can greatly influence this metric, but in no way indicates how vulnerable an application is (see sidebar in book for explanation).
  - This metric can be used to help measure the amount of work for a complete review and prioritizing remediation efforts when comparing modules and/or projects.
- Break down results by category to determine what types of problems are dominating and might require training of developers to help prevent.

Static Analysis Metrics$_3$

- Monitor trends over time.
  - Sharp increases in number of issues deserves attention.
  - Correlate increases with injection of new features or analysis of legacy code that hasn't previously been security-focused.
- Process Metrics
  - Vulnerability dwell: after an auditor identifies a vulnerability, how long, on average, does it take for the programmers to make a fix.
  - Track percent of issues reviewed and addressed. Help decide when audits should occur.

Conclusion

- Code review should be part of the software security process
- Static analysis tools can
  - Help codify best practices,
  - catch common mistakes, and
  - generally make the security process more efficient and consistent.
- Process consists of four steps:
  - defining goals,
  - running tools,
  - reviewing the code, and
  - making fixes