CIS 418/518 – Secure Software Engineering
Secure Coding with Static Analysis

Jagadeesh Nandigam

School of Computing
Grand Valley State University
nandigaj@gvsu.edu

Outline

1. Static Analysis: What? and Why?
2. Why Some Commonly Used Approaches to Software Security Fail
3. Static Analysis in the Big Picture
4. Coding Errors and Security Problems
5. Capabilities and Limitations of Static Analysis (Tools)
6. Categories of Static Analysis Tools
7. Static Analysis as Part of Code Review Process

Static Analysis: What? and Why?

- Static analysis refers to any process for assessing code without executing it.
- A static analysis tool can explore a large number of "what if" scenarios without having to go through all the computations necessary to execute the code for all the scenarios.
- Static analysis tools are well suited for identifying security defects.
- One of software security's most important practices ⇒ Code review with a static analysis tool

Why Some Commonly Used Approaches to Software Security Fail

- Defensive programming is not enough
  - Defensive programming, although a step in the right direction, does not guarantee secure software.
  - Defensive programming tries to compensate for typical kinds of accidents.
  - Software security is about creating programs that behave correctly even in the presence of malicious behavior.
  - Historically, programmers have not been trained to consider the interests or capabilities of an adversary.

Why Some Commonly Used Approaches to Software Security Fail

Consider the following C function that prints a message to a specified file descriptor without performing any error checking:

```
void printMsg(FILE *file, char *msg) {
   fprintf(file, msg);
}

// function programmed defensively
void printMsg(FILE *file, char *msg) {
   if (file == NULL) {
      logError("attempt to print message to null file");
   } else if (msg == NULL) {
      logError("attempt to print null message");
   } else {
      fprintf(file, msg);
   }
}
```

Why Some Commonly Used Approaches to Software Security Fail

Here is a version of printMsg() that a security-conscious programmer would write:

```
void printMsg(FILE *file, char *msg) {
   if (file == NULL) {
      logError("attempt to print message to null file");
   } else if (msg == NULL) {
      logError("attempt to print null message");
   } else {
      fprintf(file, "%.128s", msg);
   }
}
```

Why Some Commonly Used Approaches to Software Security Fail

- Security features != secure features
  - For a program to be secure, **all portions** of the program must be secure, not just the bits that explicitly address security.
  - In many cases, security failings are not related to security features at all.

Why Some Commonly Used Approaches to Software Security Fail

- The quality fallacy
  - It is almost imposisble to improve software security merely by improving quality assurance (through dynamic and functionality testing).
  - Functionality testing works well for making sure that typical users with typical needs will be happy, but it just won't work for finding security defects that aren't related to security features.
  - Security problems are frequently "unintended functionality" that causes the program to be insecure.
  - Some organizations attempt to overcome the lack of focus on security by mandating late-in-the-game practices like *penetration testing* and *fuzzing*.

Static Analysis in the Big Picture

- "Treating the symptom" vs. "Treating the cause"

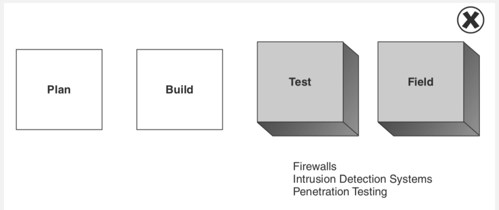


Figure 1.2 Treating the symptom: Focusing on security after the software is built is the wrong thing to do.

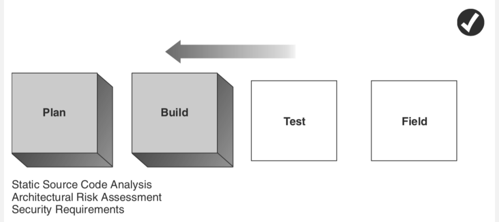
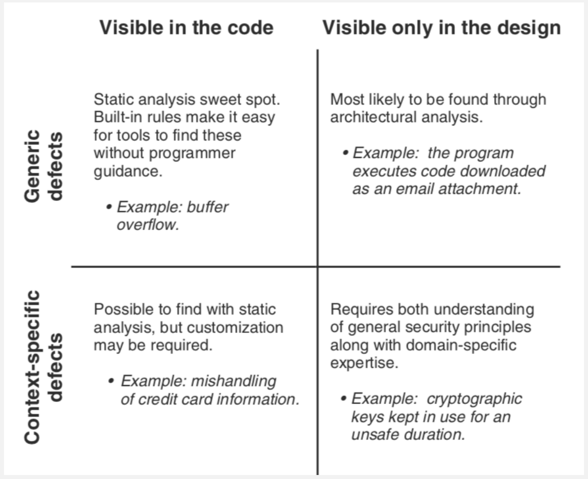


Figure 1.3 Treating the cause: Focusing on security early, with activities centered on the way the software is built.

Using Static Analysis to Find Defects

- The best way to find a particular defect (bug or flaw) depends on whether it is generic or context specific, and whether it is visible in the code or only in the design.



	Visible in the code	Visible only in the design
Generic defects	Static analysis sweet spot. Built-in rules make it easy for tools to find these without programmer guidance. • *Example: buffer overflow.*	Most likely to be found through architectural analysis. • *Example: the program executes code downloaded as an email attachment.*
Context-specific defects	Possible to find with static analysis, but customization may be required. • *Example: mishandling of credit card information.*	Requires both understanding of general security principles along with domain-specific expertise. • *Example: cryptographic keys kept in use for an unsafe duration.*

Coding Errors and Security Problems

- Kinds of coding errors that are likely to lead to security problems:
  - Input validation and representation
  - API abuse
  - Security features
  - Time and state
  - Error handling
  - Code quality
  - Encapsulation (drawing strong boundaries)
  - Environment (influence of outside factors)

Capabilities and Limitations of Static Analysis (Tools)

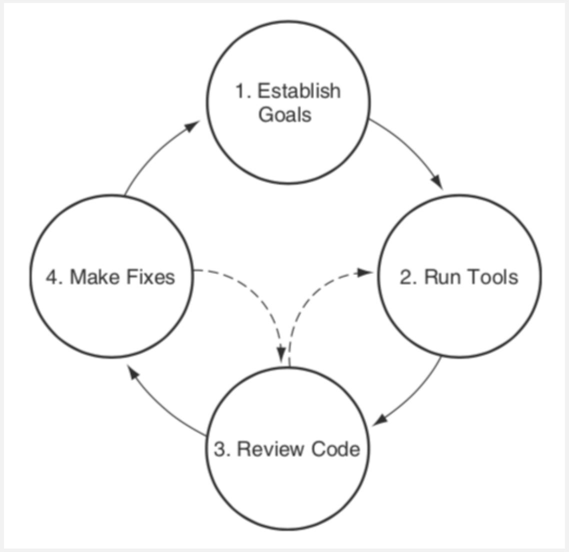
- Capabilities
  - Static analysis tools apply checks thoroughly and consitently without any bias.
  - By examining the code itself, static analysis tools can often point to the root cause of a security problem, not just one of its symptoms.
  - Static analysis can find errors early in development, even before the program is run for the first time.
- Limitations
  - Static analysis tools that target security may produce too much noise in terms of *false positives* or *false alarms*.
  - A false positive is a problem reported in a program when no problem actually exists.
  - From a security perspective, *false negatives* are much worse. With a false negative, a problem exists in the program, but the tool does not report it.
  - The penalty for a false negative is much greater.
  - Since the penalty for overlooked security bugs is high, security tools usually produce more false positives to minimize false negatives.

Categories of Static Analysis Tools

There are many kinds of static analysis tools, each with different goals:

- Type checking
- Style checking
- Program understanding (reverse engineering)
- Program verification
- Property checking
- Bug finding
- Security review

Static Analysis as Part of Code Review Process



Avoid Exploitability Trap During Code Review

- Security review of code should not be about creating flashy exploits for problems found.
- Don’t get pulled down into exploit development.
- When a developer says, “I won‘t fix that unless you can prove it‘s exploitable,” you’re looking at the exploitability trap.
- The exploitability trap is dangerous for two reasons:
  - Developing exploits is time consuming.
  - Developing exploits is a skill unto itself.
- Instead, focus on getting the security bugs fixed!!

References

- Brian Chess and Jacob West, *Secure Programming with Static Analysis*, Chapters 1 – 3, Addison Wesley.