Increase Adoption of Secure Coding Standards
Dan Plakosh

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Outline

- **Fundamentals: Prescriptive Rules**
  - Maintain the C rules
  - New computation models: threads
  - Major language updates: C++

- **Reducing friction of adoption**
  - Improving analyst productivity
  - Immediate feedback and correction
  - Catching more violations

- **Summary**
Fundamentals: C Coding Rules

- Specific, prescriptive advice for programmers, checkers and IDEs
- Collected wisdom of programmers and tools vendors
  - Fed by community wiki started in Spring 2006
  - 1,576 registered contributors
  - Basis for ISO Standard
- Continuously updated to reflect best practices and language evolution
Fundamentals: Expanding Coverage

New computation model: C threads

- 9 unspecified behaviors representing programming weaknesses in two broad categories
  - Inter-thread communication
  - Thread-specific storage

Example: the `tss_create` function which creates thread-specific storage and assigns a destructor but does not specify when the destructor is called.

Major language updates for C++

- 24 new rules in FY15 specifying C++ weaknesses
- 60 existing C++ rules updated in FY15

New and updated rules published on

http://www.cert.org/secure-coding/publications/secure-coding-enenewsletter.cfm
Adoption: Improving Analyst Productivity

Improve expert review productivity by focusing on high priority violations

Filter select secure coding rule violations

- Eliminate irrelevant diagnostics
- Convert to common CERT Secure Coding rule labeling

Provide single view into code and all diagnostics

Maintain record of decisions
Adoption: Immediate Feedback

Moving rules into IDE improves application of secure coding

• Early feedback corrects errors on introduction
• Exceptions are understood in context
• Exceptions can be marked as resolved to eliminate redundant consideration

Clang static analyzer (C based languages)

• Widely used open source front end for popular compilers and IDEs
• Checkers available now in “Top-of-Tree” by early adopters
• Expect to be generally available in Clang’s yearly release

FindBugs (Java)

• Integrated into Eclipse and Jdeveloper
Adoption: Catching More Violations Thru Checkers

- Checking C/C++ rule violations
  - Exception
  - Function return
  - Evaluation ordering / side effects
  - Constructor
  - Assertion

  Example:
  ```c++
  int a = 14;
  int b = sizeof(a++);
  std::cout << a << " , " << b << std::endl;
  a is still 14 after b has been initialized
  ```

- Checking Java rule violations
  - Override
  - I/O

  Example:
  ```java
  byte data;
  while ((data = (byte) in.read()) != -1) {
    // ...
  }
  ```

Increase adoption through automated checkers of rule violations
Summary

• Maintained C Coding rules - updated to reflect best practices and language evolution
• Developed 25 new rules for C++ and updated 60 existing C++ rules
• Developed a web application to improve analyst productivity
• Introducing checking earlier in the development process
  ✓ developed checkers for clang and FindBugs