Sticking to the Facts II: Scientific Study of Static Analysis Tools

SATE IV
Workshop
March 29, 2012

Center for Assured Software
National Security Agency
cas@nsa.gov

Agenda

- Background and Purpose
- NSA CAS Methodology Review
- 2011 Results/Trends
  - Data Analysis and Visualizations

Center for Assured Software

Mission: To positively influence the design, implementation, and acquisition of Department of Defense (DoD) systems to increase the degree of confidence that software used within the DoD's critical systems is free from intentional and unintentional exploitable vulnerabilities

NSA CAS Methodology – A Review

Study Process Overview

1. Generate test cases (Juliet Test Suite)
2. Analyze test cases per tool
3. Score results
4. Group test cases into Weakness Classes
5. Calculate statistics by each Weakness Class

CAS Test Cases

- Artificial pieces of code developed to test software analysis tools
- Mapped to CWEs
- In general, each test case contains:
  - One flawed construct – “bad”
  - One or more non-flawed constructs that “fix” the flawed construct – “good”

Example of a Test Case

```
void CWE467_Use_of_sizeof_on_Pointer_Type__double_01_bad()
{
    double * data;
    ...
    /* FLAW: Using sizeof the pointer and not the data type in
malloc() */
    data = (double *)malloc(sizeof(data));

}

static void goodG2B()
{
    double * data;
    ...
    /* FIX: Using sizeof the data type in malloc() */
    data = (double *)malloc(sizeof(*data));

}
```

Advantages / Limitations of Test Cases

- Advantages
  - Control over the breadth of flaws and non-flaws covered
  - Control over where flaws and non-flaws occur
  - Control over data and control flows used

- Limitations
  - Simpler than natural code
  - All flaws represented equally
  - Ratio of flaws and non-flaws likely much different than in natural code

Weakness Classes

Weakness Class	Example Weakness (CWE Entry)
Authentication and Access Control	CWE-620: Unverified Password Change
Buffer Handling	CWE-121: Stack-based Buffer Overflow
Code Quality	CWE-561: Dead Code
Control Flow Management	CWE-362: Race Condition
Encryption and Randomness	CWE-328: Reversible One-Way Hash
Error Handling	CWE-252: Unchecked Return Value
File Handling	CWE-23: Relative Path Traversal
Information Leaks	CWE-534: Information Leak Through Debug Log Files
Initialization and Shutdown	CWE-415: Double Free
Injection	CWE-89: SQL Injection
Miscellaneous	CWE-480: Use of Incorrect Operator
Number Handling	CWE-369: Divide by Zero
Pointer and Reference Handling	CWE-476: Null Pointer Dereference

Scoring

- CAS is concerned with two things:
  - What flaws does the tool report? (Recall)
  - What non-flaws does the tool incorrectly report as a flaw? (Precision)

Precision

- Fraction of results from tool that were “correct”

$$Precision = \frac{\#TP}{\#TP + \#FP}$$

- Same as “True Positive Rate”
- Complement of “False Positive Rate”

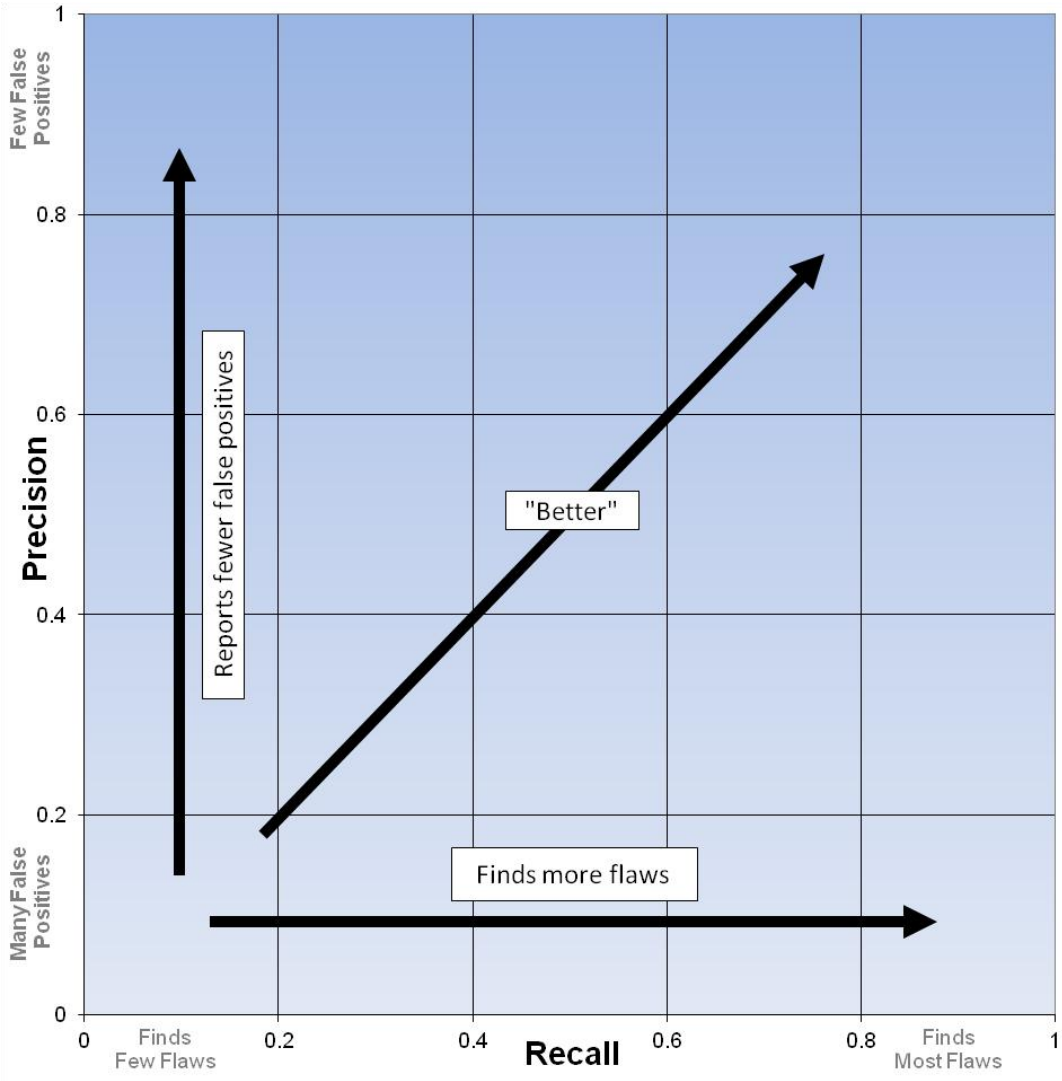
Recall

- Fraction of flaws that a tool correctly reported

$$Recall = \frac{\#TP}{\#TP + \#FN}$$

- Also known as “Sensitivity” or “Soundness”

Precision-Recall Graph



Precision and Recall Are Not Enough

- Precision and Recall don't tell whole story
- Unsophisticated "grep-like" tool can get:
  - Recall: 1
  - Precision: 0.5
  - Doesn't accurately reflect that tool is noisy
- Limitation of CAS test cases
  - Typically 1 or 2 non-flaws for each flaw

Discrimination

- A “Discrimination” occurs when a tool:
  - Correctly reports the flaw
  - Does not report the non-flaw
- Each tool gets 0 or 1 discrimination for each test case

Discrimination Rate

- Discrimination Rate is the fraction of test cases where a tool reported discriminations

$$Discrimination\ Rate = \frac{\#\,Discriminations}{\#\,Flaws}$$

- Discrimination Rate ≤ Recall
  - Every True Positive “counts” toward Recall, but not necessarily toward Discrimination Rate

2011 Methodology Changes

- New flaws as well as data and control flow variants were added
  - Java Test Cases increased by 74%
  - C/C++ Test Cases increased by 26%
- Test cases were enhanced
- Analysis was improved
  - Recall calculation
  - Test case weighting
- Tool configurations

2011 Methodology Changes (cont.)

Weakness Class	Example Weakness (CWE Entry)
Authentication and Access Control	CWE-620: Unverified Password Change
Buffer Handling	CWE-121: Stack-based Buffer Overflow
Code Quality	CWE-561: Dead Code
Control Flow Management	CWE-362: Race Condition
Encryption and Randomness	CWE-328: Reversible One-Way Hash
Error Handling	CWE-252: Unchecked Return Value
File Handling	CWE-23: Relative Path Traversal
Information Leaks	CWE-534: Information Leak Through Debug Log Files
Initialization and Shutdown	CWE-415: Double Free
Injection	CWE-89: SQL Injection
Malicious Logic	CWE-506: Embedded Malicious Code
Miscellaneous	CWE-480: Use of Incorrect Operator
Number Handling	CWE-369: Divide by Zero
Pointer and Reference Handling	CWE-476: Null Pointer Dereference

2011 Study Results and Trends

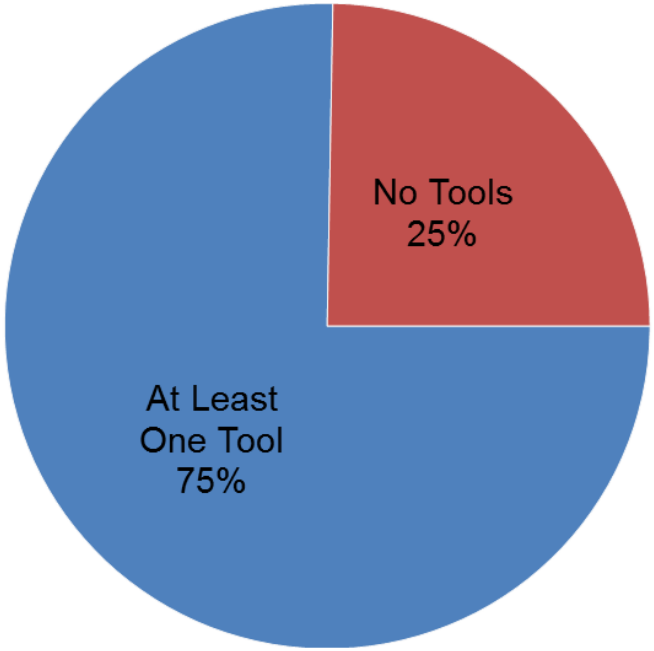
C/C++

- Tools Studied
  - 8 commercial
  - 1 open source

	CWEs Covered	Flaw Types	Test Cases	Lines of Code
2010	116	1,432	45,324	6,338,548
2011	119	1,489	57,099	8,375,604
Diff	+ 2.6%	+ 4.0 %	+ 26.0%	+ 32.1%

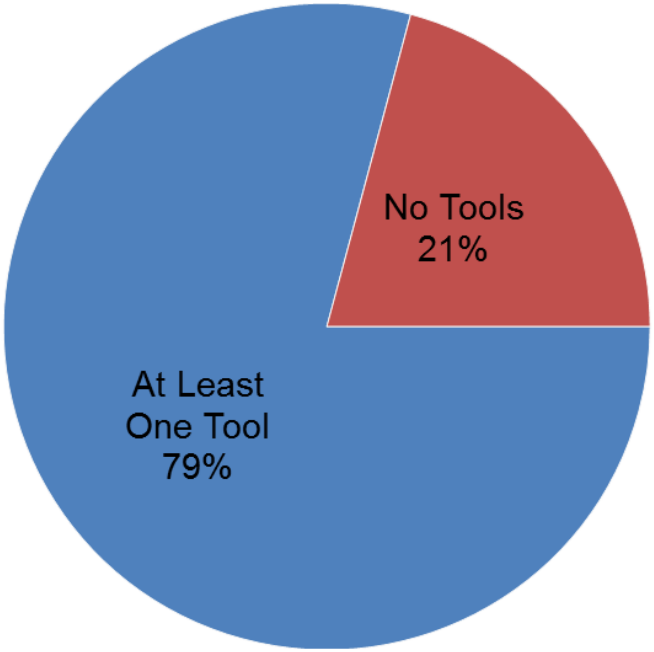
Test Case Coverage C/C++

C/C++ Test Cases (2010)



- Seven tools
- 45,324 Test Cases

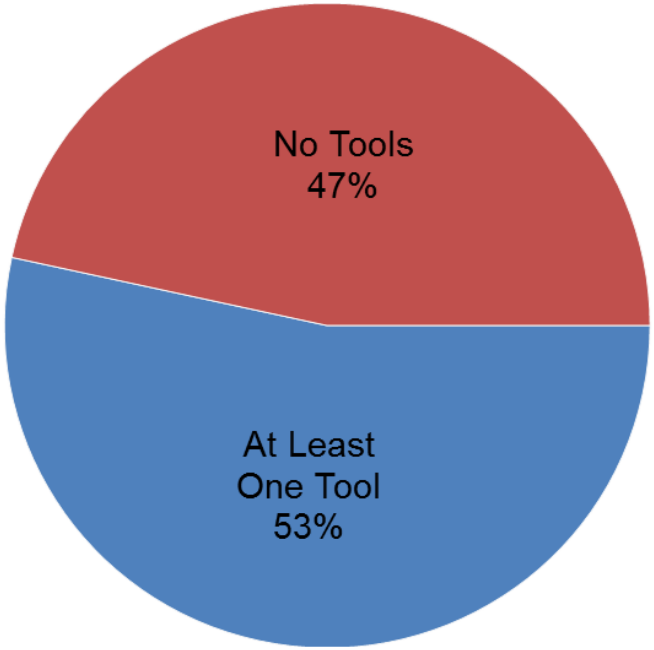
C/C++ Test Cases (2011)



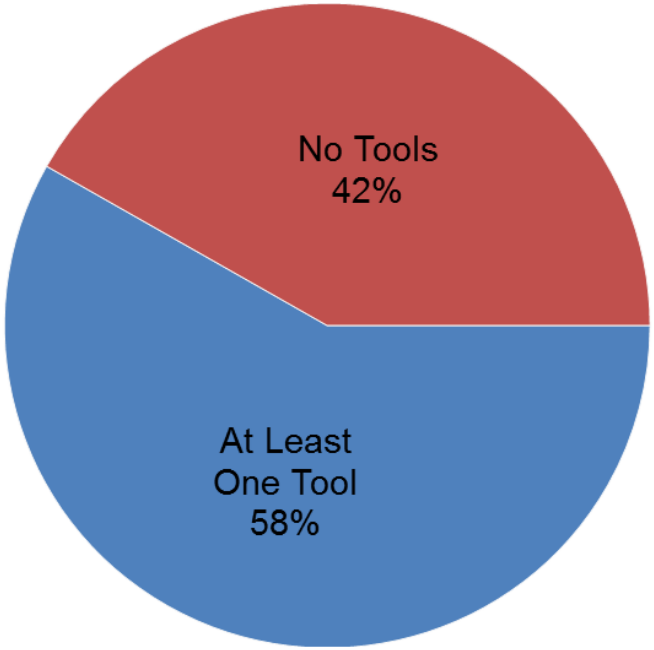
- Nine tools
- 57,099 Test Cases

Test Case Discriminated – C/C++

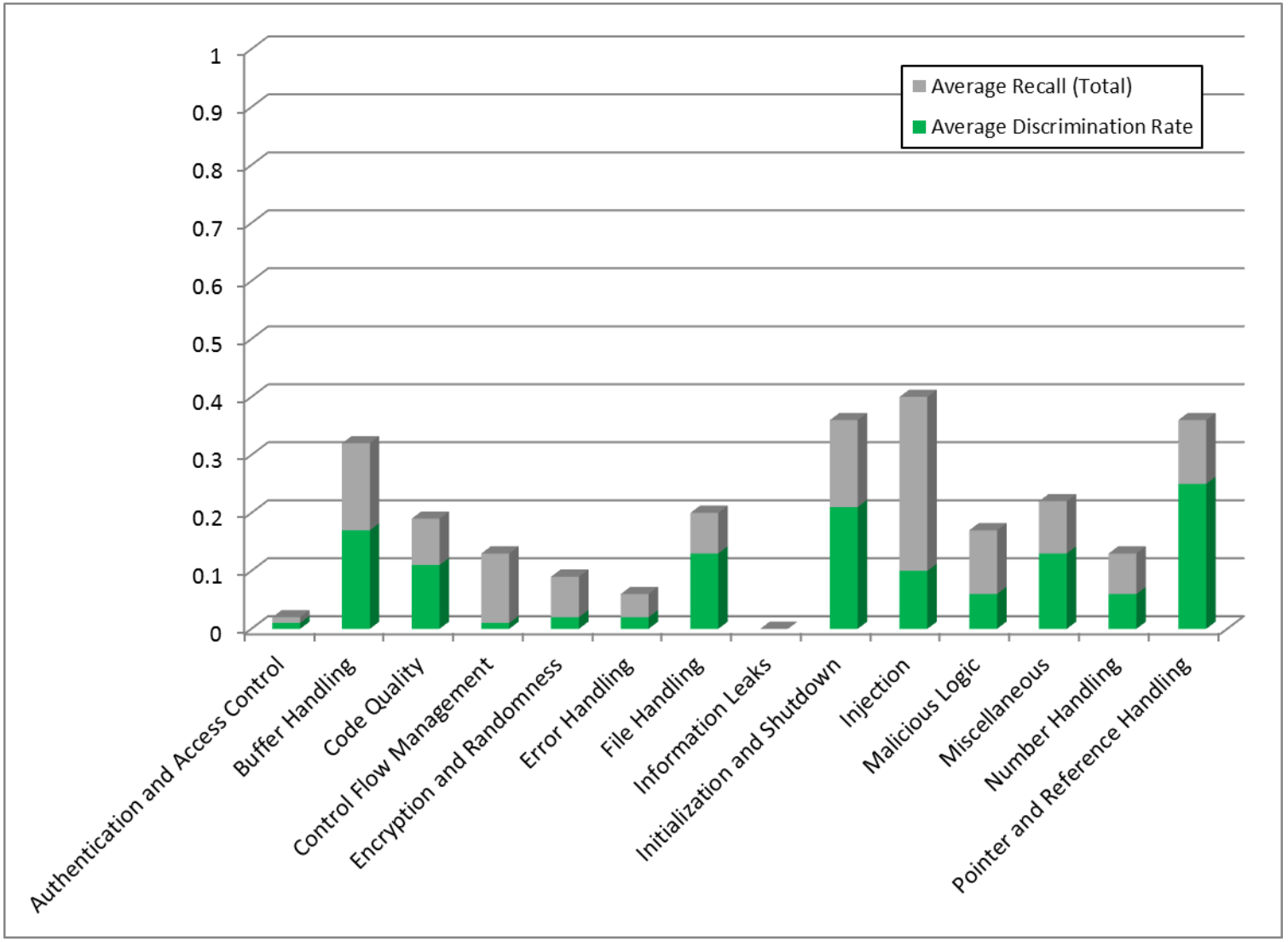
C/C++ Test Cases (2010)



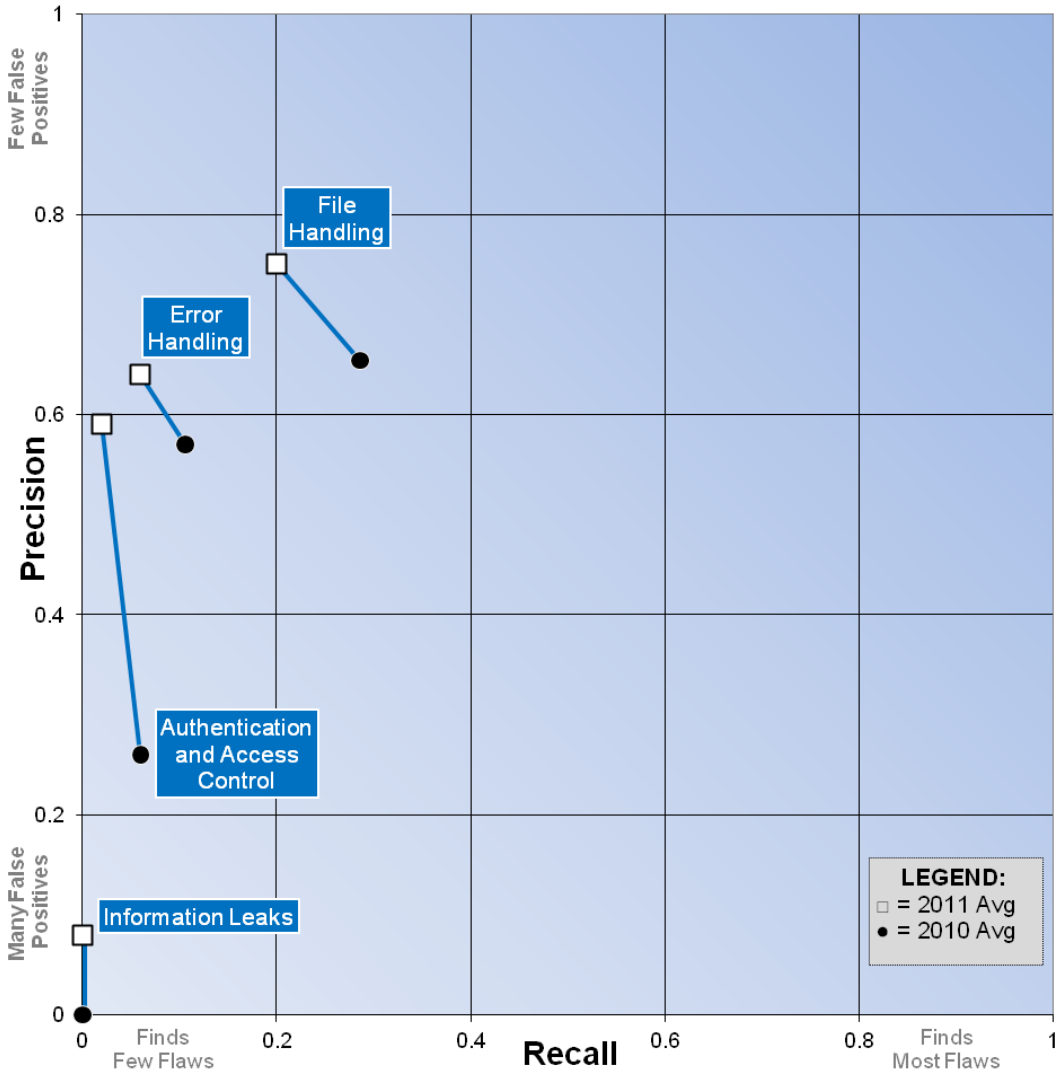
C/C++ Test Cases (2011)



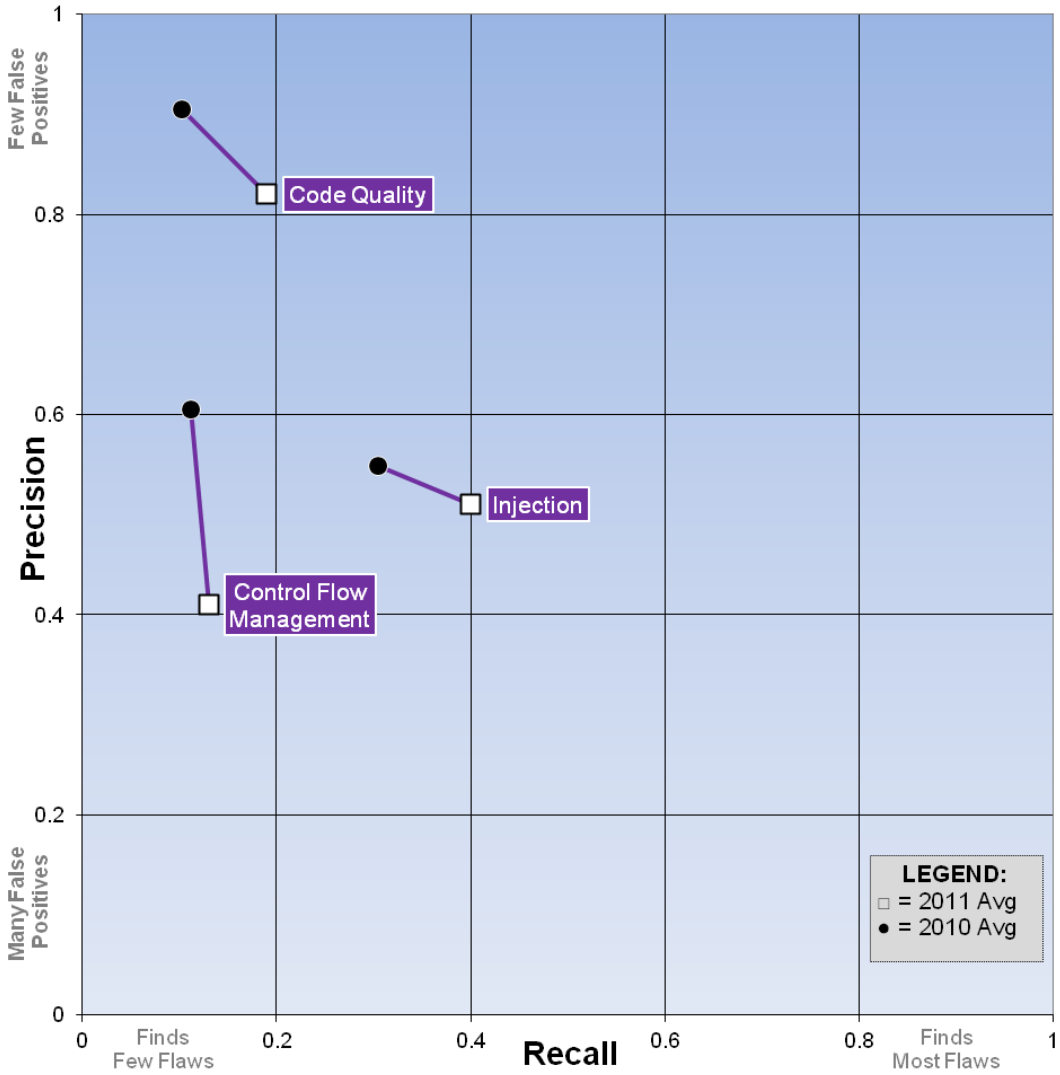
Test Case Coverage and DR – C/C++ 2011



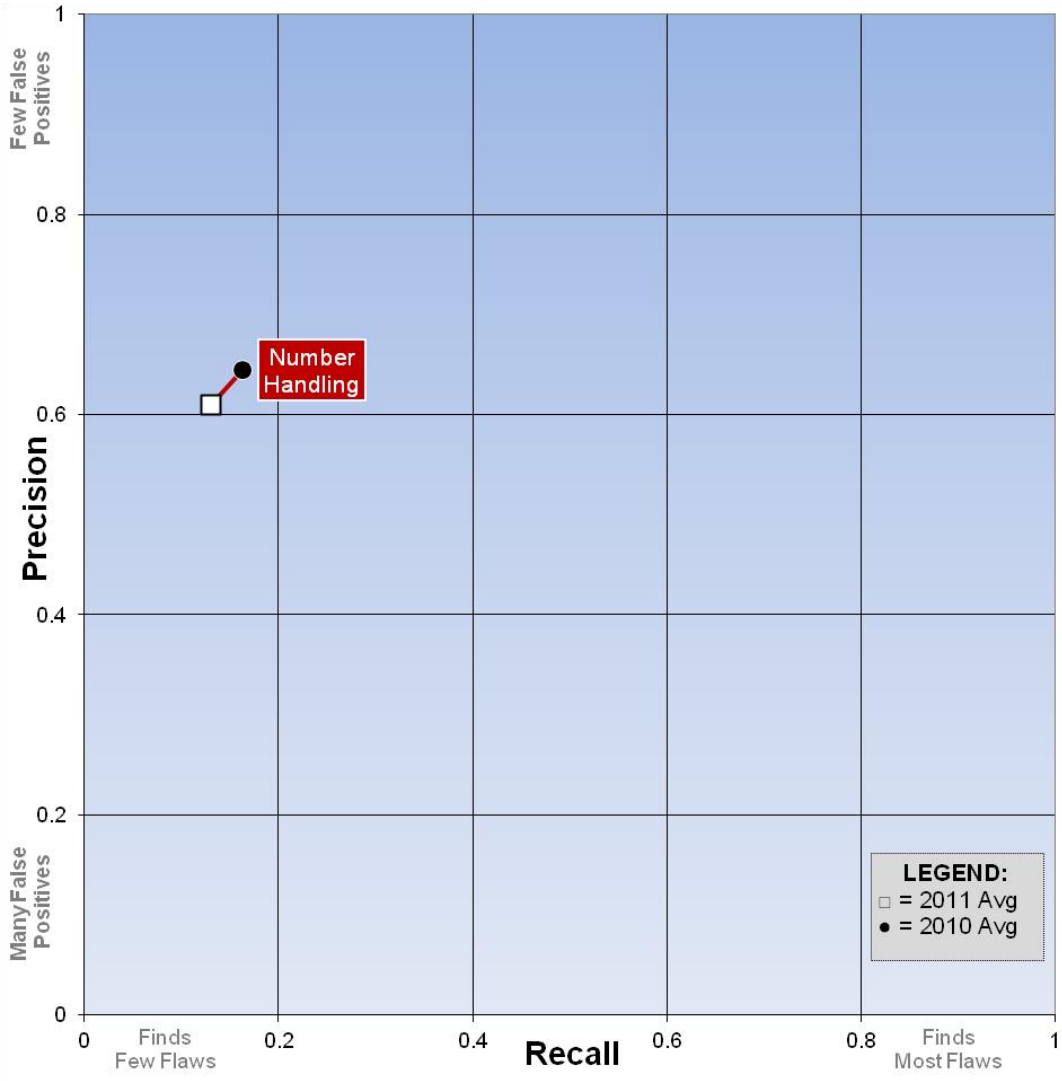
Improved Precision – C/C++



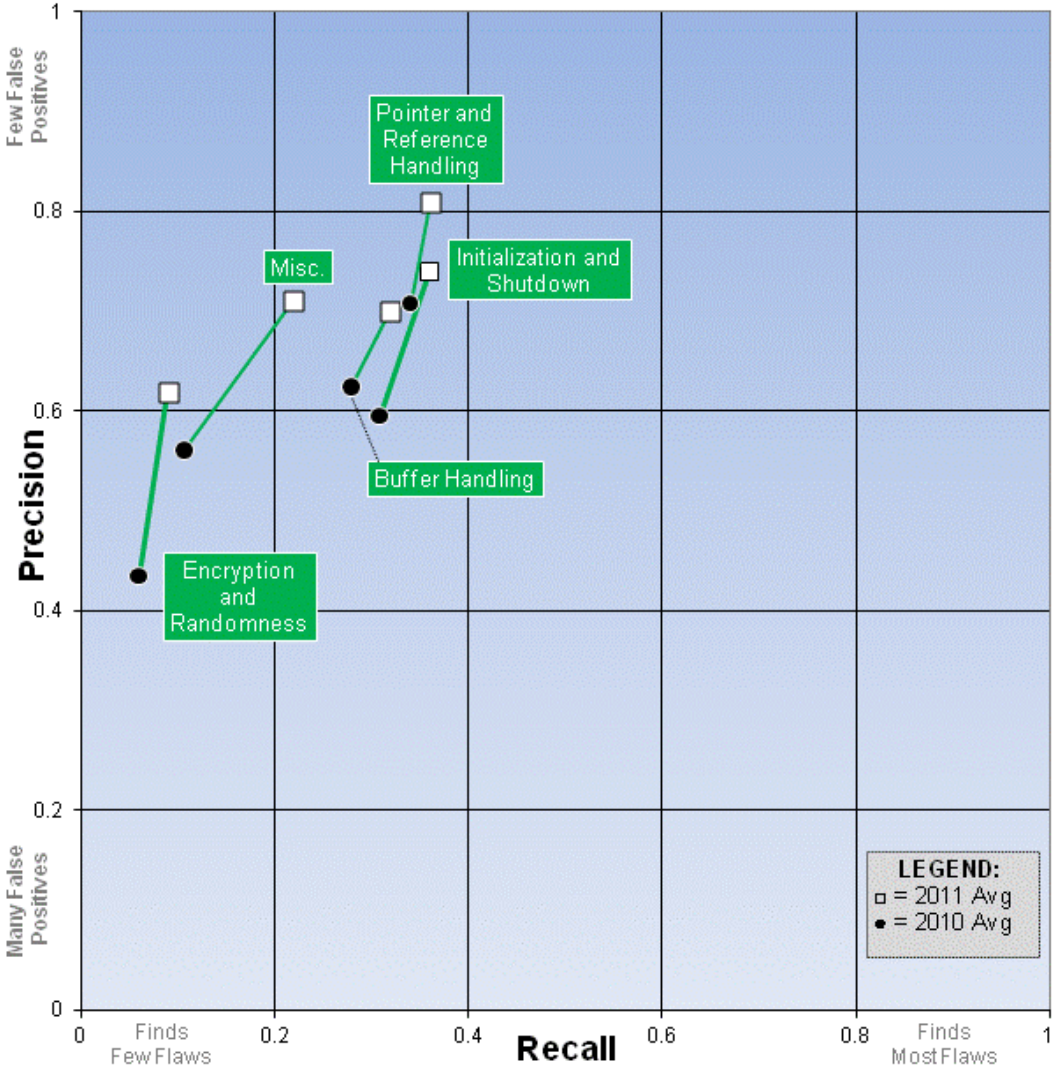
Improved Recall – C/C++



Precision and Recall Less - C/C++



Precision and Recall Improved - C/C++



Tool Combination – C/C++

	Tool #1		Tool #2	
	Disc. Rate	Recall	Disc. Rate	Recall
Tool #1	.24	.40	.51	.67
Tool #2	.51	.67	.38	.57

2011 C/C++ Conclusions

- Tools Strongest in:
  - Pointer and Reference Handling
  - Initialization and Shutdown
  - Buffer Handling
- Tools Weakest in:
  - Information Leaks
  - Authentication and Access Control
  - Error Handling

2011 C/C++ Conclusions (cont.)

- Reported flaws in approximately 11 of the 14 (79%) Weakness Classes
- Reported approximately 22% of the flaws on Weakness Classes they covered
- Flaws in approximately 21% of the test cases were not reported by any of the tools
- There were 18 test cases in which all of the tools correctly found the flaw

Open Source vs. Commercial Tools – C/C++

- Did not perform the strongest in any of the Weakness Classes
- Stronger than at least 1 commercial tool in 6 Weakness Classes
- In 4 Weakness Classes, was the weakest tool

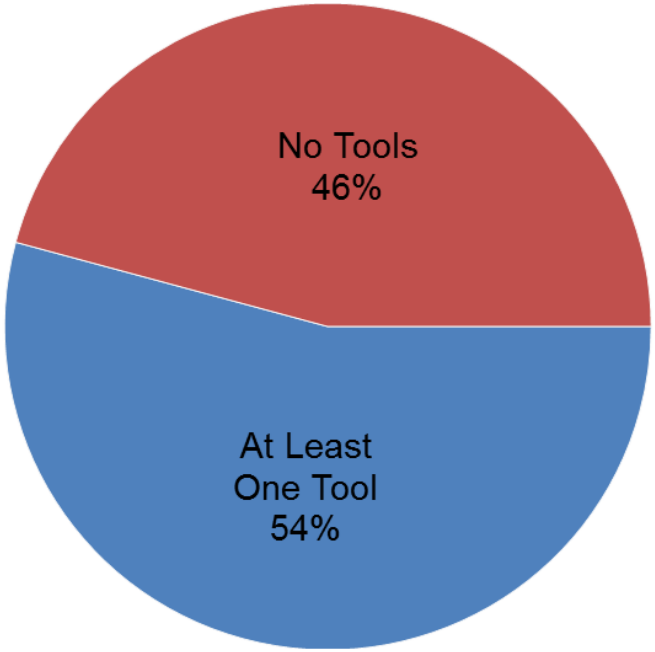
Java

- Tools Studied
  - 7 commercial
  - 2 open source

	CWEs Covered	Flaw Types	Test Cases	Lines of Code
2010	106	527	13,801	3,238,667
2011	113	751	23,957	4,712,718
Diff	+ 6.6%	+ 42.5%	+ 73.6%	+ 45.4%

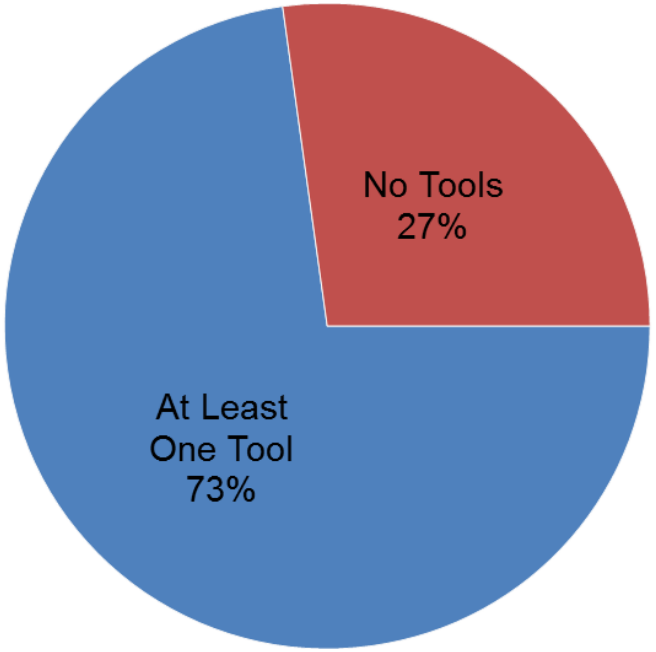
Test Case Coverage Java

Java Test Cases (2010)



- Seven tools
- 13,801 Test Cases

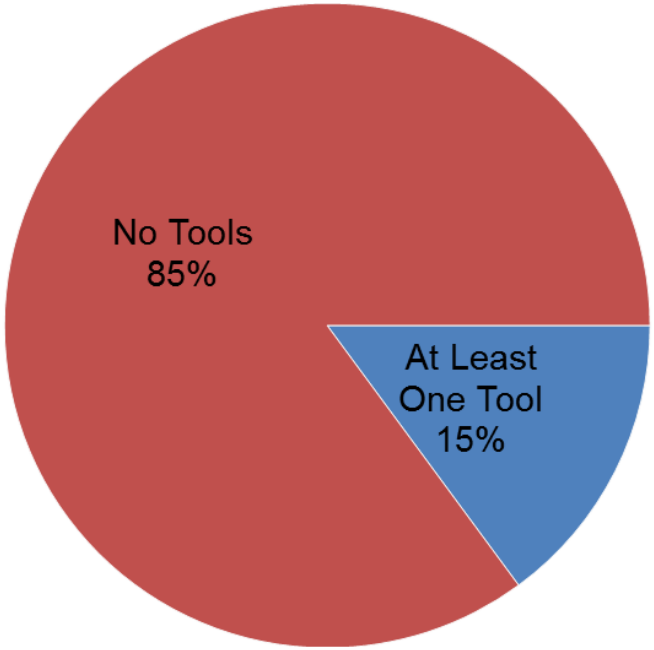
Java Test Cases (2011)



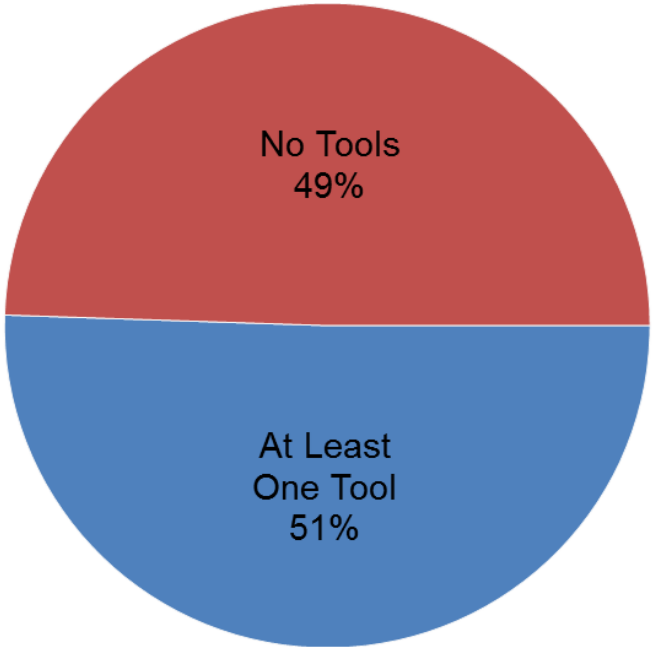
- Nine tools
- 23,957 Test Cases

Test Case Discriminated – Java

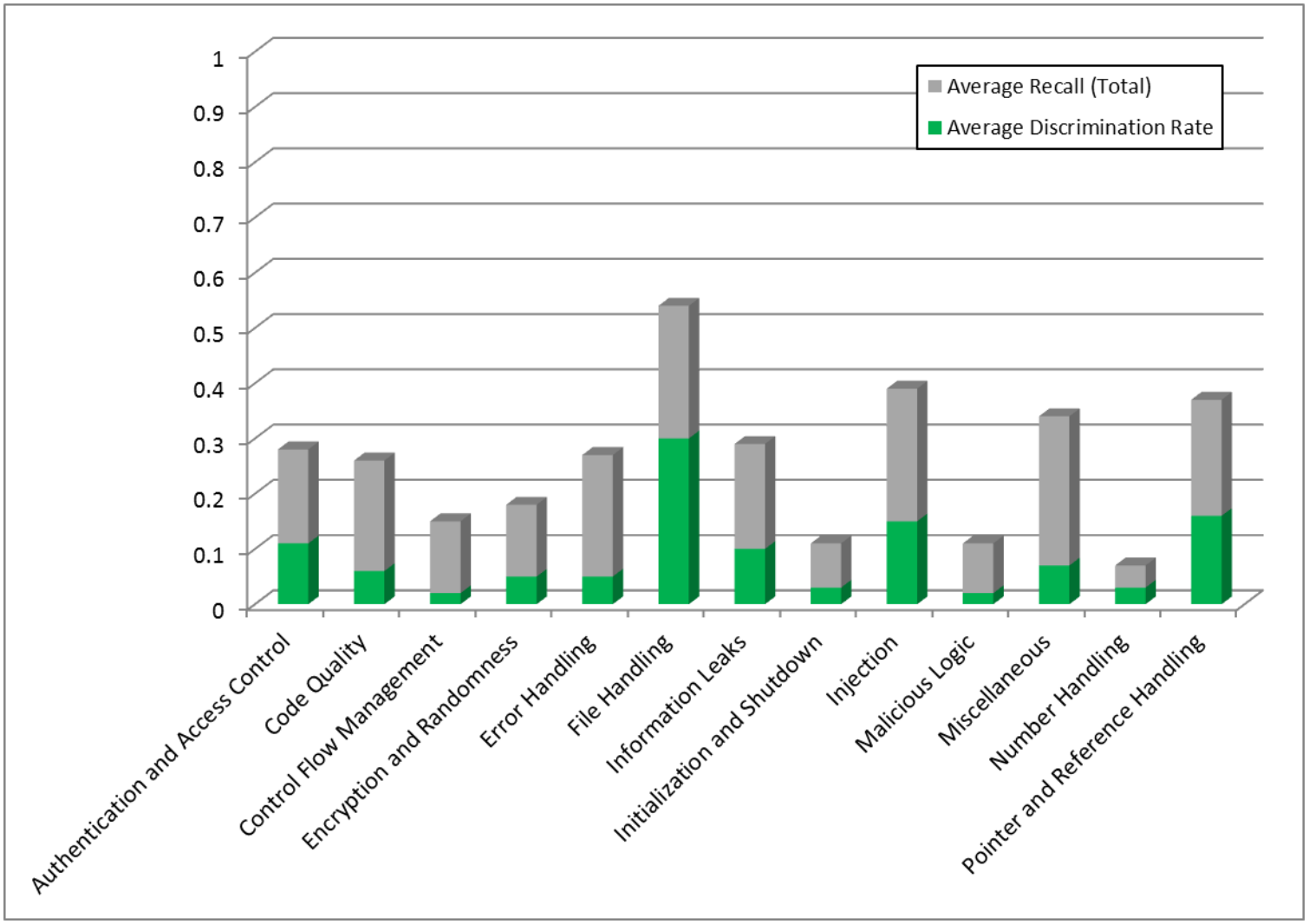
Java Test Cases (2010)



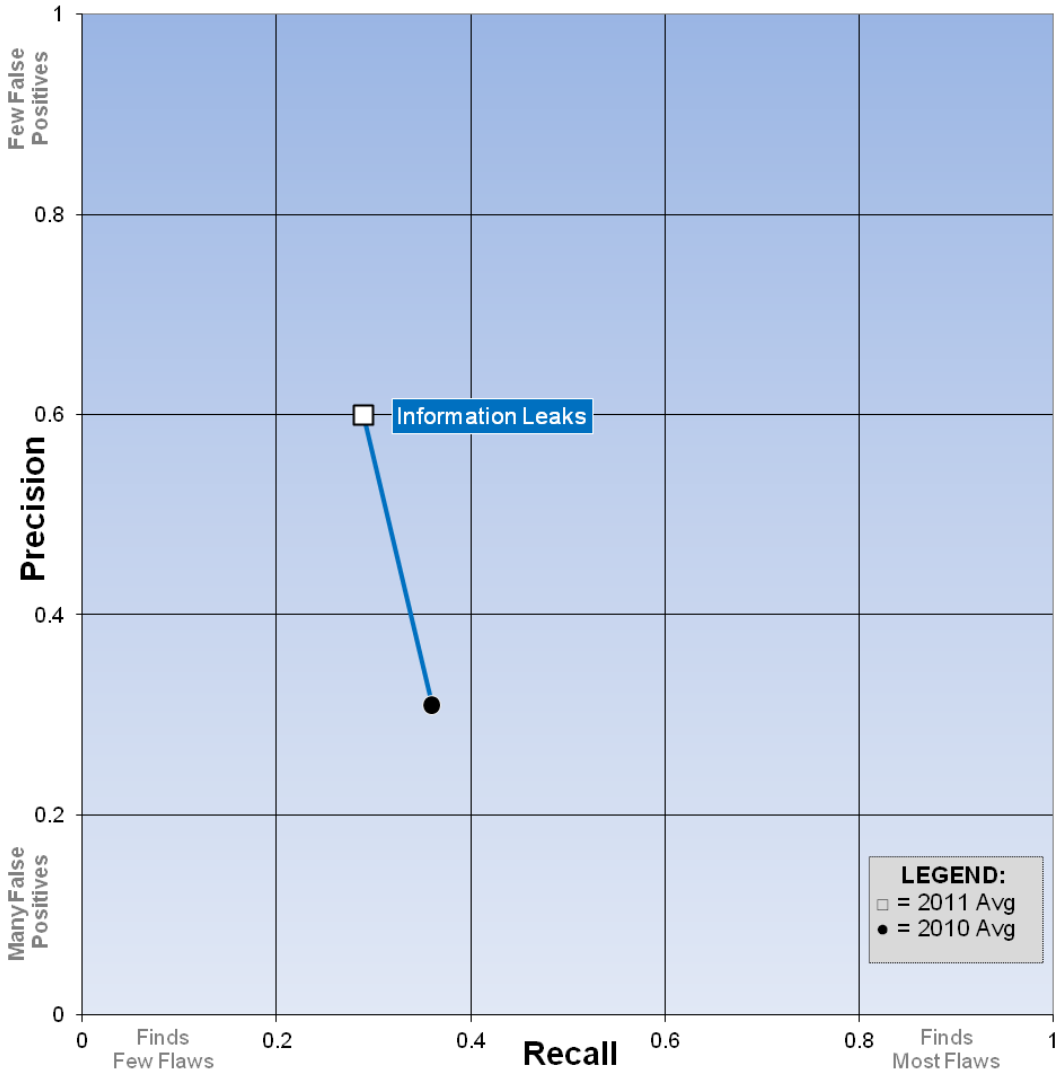
Java Test Cases (2011)



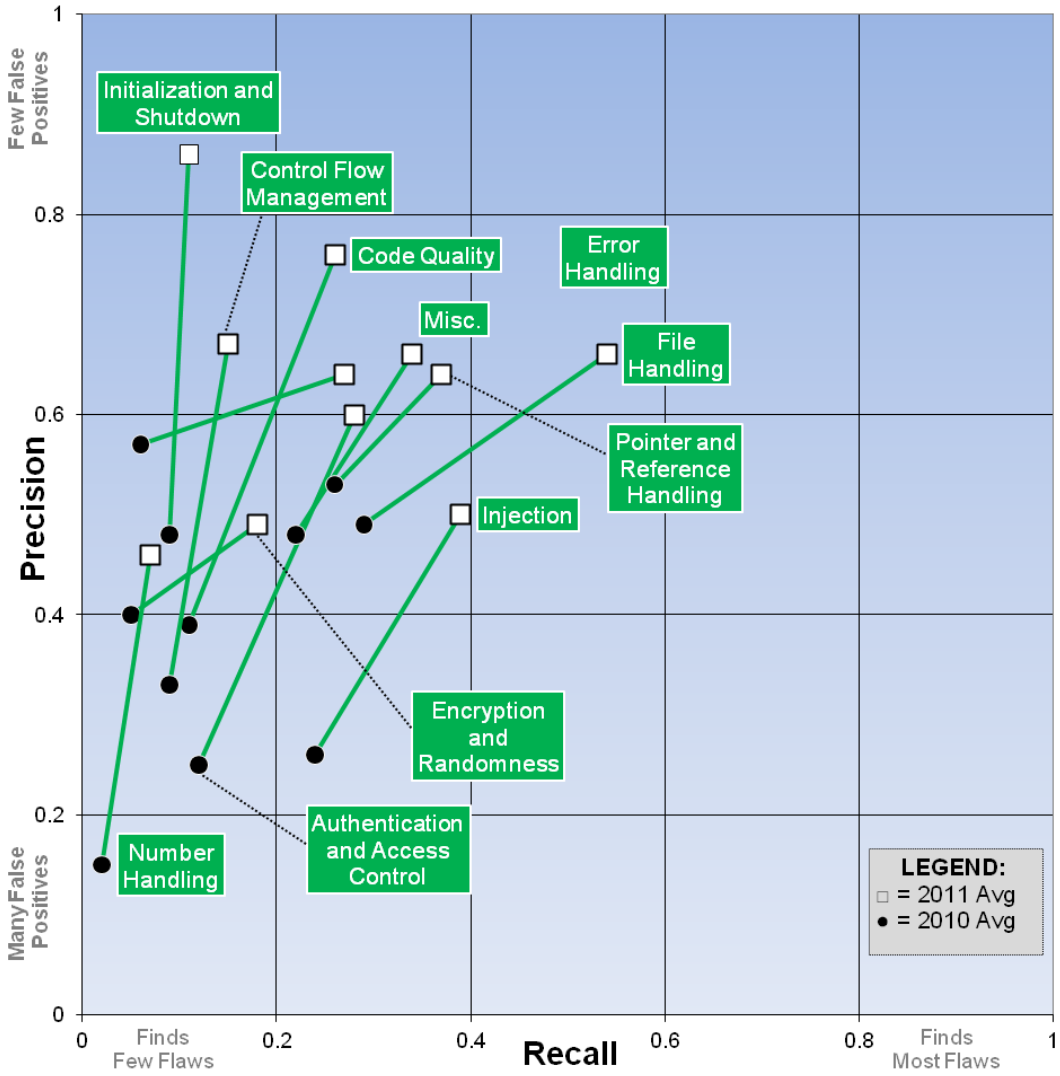
Test Case Coverage and DR – Java 2011



Precision Improved - Java



Precision and Recall Improved – Java



Tool Combination – Java

	Tool #1		Tool #2	
	Disc. Rate	Recall	Disc. Rate	Recall
Tool #1	.30	.57	.45	.68
Tool #2	.45	.68	.27	.43

Java Conclusions

- Tools Strongest in:
  - File Handling
  - Pointer and Reference Handling
- Tools Weakest in:
  - Number Handling
  - Malicious Logic
  - Initialization and Shutdown

Java Conclusions (cont.)

- Reported flaws in approximately 10 of the 13 (77%) Weakness Classes
- Reported approximately 28% of the flaws on Weakness Classes they covered
- Flaws in approximately 27% of the test cases were not reported by any of the tools
- There were no test cases in which all of the tools correctly found the flaw

Open Source vs. Commercial Tools Java

- None of the open source tools performed the strongest in any of the Weakness Classes
- At least 1 open source tool was stronger than at least 1 commercial tool in 7 Weakness Classes
- In 3 Weakness Classes, 1 open source tool was ranked in the top 3
- In four Weakness Classes, the open source tools were the weakest tools

2011 Study Conclusions

- Tools are not interchangeable
- Different tools had different strengths, even different by language
- None of the tools performed well across all Weakness Classes
- Complementary tools can be combined to achieve better results

Can Tools Be Improved?

- Goodness of code
  - Report proper coding techniques
  - Aids in overall analysis of code
- Standardized Output
  - Flaw location
  - Results format
  - Flaw Naming convention

Questions?

- Juliet Test Suite v1.1 and Methodology Report (will be) located at http://samate.nist.gov/SRD/testsuite.php

- Contact Center for Assured Software at CAS@nsa.gov