Sticking to the Facts

Scientific Study of Static Analysis Tools

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

Agenda

- Background and Purpose
- Test Cases
- Scoring Tool Results
- Data Analysis and Visualizations
- 2010 Study Conclusions
  - Open Source vs. Commercial Tools
- 2011 Study Plans

Center for Assured Software

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
- Strategy:
  - Assess and Understand currently available Software Assurance (SwA) Techniques and Technology
  - Influence (Outreach to) the DoD, US Government, Private Sector and Academia on SwA policy, development, deployment and research
  - Apply and implement current SwA Tools, Techniques and methods to DoD and Intelligence Community clients

Center for Assured Software

- CAS Technology Focus Area
  - Encourages the appropriate use of automation to measure assurance properties of software
  - “Let the code speak”
  - Spends a significant amount of time looking for new software assurance tools, testing tools and reporting on tools to support software assurance analysis

What is Static Analysis?

Static Analysis

- Static analysis of software is a method of examining software without executing it
- Analyzes software itself, not documentation
  - Often done on software's source code
  - Can be done on compiled binaries
- Applicable to all software types and languages
  - Tools focus on more popular types and languages
- Also known as:
  - "Static Code Analysis"
  - "Static Program Analysis"
  - "Source Code Analysis"

Static Analysis Tools

- Static analysis tools automate the process of doing static analysis
- Commercial and no cost tools are available
- Vary widely in capabilities, features, and cost
- This presentation covers tools that identify and report issues in the software
- Also known as:
  - “Code Weakness Analysis Tools”
  - “Static Application Security Testing Tools”

Benefits of Static Analysis Tools

- Identify errors in software (bugs)
  - Including security issues
  - Good at finding some types of issues
- Analyzes all parts of the software
  - Unlike external testing (dynamic analysis) which only examines the code paths exercised
- Automated, scalable, repeatable
  - Unlike manual code review
  - Can be used early and often

Limitations of Static Analysis Tools

- Most do not report positive properties (or lack thereof)
- May report false positives (reports of an issue where none exists) along with real results
- May report issues that are not important to you or your software
- Cannot always definitively report issues
  - Sometimes report only that an issue may be present at a location
  - Needs confirmation by a human

Limitations of Static Analysis Tools

- Do not cover all flaw types
  - Better at implementation issues vs. design issues
  - Scrutinize vendor claims
- Typically miss issues (false negatives)
  - May create false sense of security
- Tool coverage is detailed in the next section

CAS 2010 Static Analysis Tool Study

Study Purpose

- Study capabilities of commercial and open source static analysis tools for C/C++ and Java
  - Identify areas in which individual tools are strong
  - Determine how tools can be combined to use strong tool(s) in each area
- Study does NOT:
  - Attempt to choose a “best” tool
  - Cover anything other than results
    - Cost, performance, ease of use, customization, etc.

Tools Studied

Tool	License Model	C/C++	Java
Tool 1	Commercial	✓	✓
Tool 2	Commercial	✓	✓
Tool 3	Commercial	✓	✓
Tool 4	Commercial	✓	✓
Tool 5	Commercial	✓	✓
Tool 6	Commercial	✓	
Tool 7	Open Source	✓	
Tool 8	Open Source		✓
Tool 9	Open Source		✓

Study Methodology Overview

- Analyze test cases with each tool in its default configuration
- Convert the results into a CAS-defined, common, Comma Separated Value (CSV) format
- Score results
  - Mark results relevant to test case as True Positives or False Positives
  - Add False Negatives
- Group test cases into “weakness classes”
- Calculate statistics for each weakness class

Differences from NIST SATE/SAMATE

- We run each tool, not the tool vendor
- We use synthetic test cases instead of natural code
- We know where all the target flaws and non-flawed constructs are intended to be
- We know what type of flaw and non-flaw each construct is intended to represent


NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA
CENTRAL SECURITY SERVICE
UNITED STATES OF AMERICA


IAD

Test Cases

CAS Test Cases

- Test cases are artificial pieces of code for testing software analysis tools
- Each test case contains:
  - One flawed construct – “bad”
  - One or more non-flawed constructs that “fix” the flawed construct – “good”
    - As much as possible, performs the same function as the flawed construct
- Test cases cover:
  - C/C++
  - Java

Example of a Test Case

```
void CWE134_Uncontrolled_Format_String__
     scanf_to_printf_01_bad()
{
  char buf[100];
  if (scanf("%99s", buf) == 1)
  {
    /* FLAW: buf (obtained from scanf) is
       passed as the format string to printf */
    printf(buf);
  }
}
```

Example of a Test Case (cont'd)

```
static void good3()
{
  char buf[100];
  if (scanf("%99s", buf) == 1)
  {
    /* FIX: Use %s as a format string and
       pass buf as an argument */
    printf("%s", buf);
  }
}
```

Advantages of Test Cases

- Control over the breadth of flaws and non-flaws covered
  - Study full range of tools' capabilities
- Control over where flaws and non-flaws occur
  - Allows for automated scoring of results
- Control over data and control flows used
  - Study depth of tools' analysis
  - Test cases for many flaw types cover
    - Simplest form of flaw
    - 18 different control flow patterns
    - 22 different data flow patterns

Limitations of Test Cases

- Simpler than natural code
  - Tools may have “better” results on test cases than on natural code
- All flaws represented equally
  - Each flaw appears one time in test cases, regardless of how common the flaw is in natural code
- Ratio of flaws and non-flaws likely much different than in natural code
  - 1 or 2 non-flaw(s) for each flaw in the test cases
  - In natural code, non-flaws are likely much more common than flaws

Test Case Scope

- Test cases are currently focused on:
  - Functions available on the underlying platform
    - Not the use of third-party libraries or frameworks
  - Platform-neutral and Windows-specific functions
    - No test cases specific to Linux, Mac OS, etc.
  - C language vs. C++
    - C++ is only used for flaw types that require it (such as leaks of memory allocated with “new”)
  - Java applications and Servlets
    - No Applets or Java Server Pages (JSPs)

2010 Test Case Statistics

	CWEs Covered	Flaw Types	Test Cases	Lines of Code
C/C++	116	1,432	45,324	6,338,548
Java	106	527	13,801	3,238,667
All Test Cases	**177**	**1,959**	**59,125**	**9,577,215**

Test Cases available as Juliet Test Suites at http://samate.nist.gov/SRD/testsuite.php

Scoring Tool Results

Scoring Tool Results

- Vast majority of tool results are automatically scored with CAS created tool based on:
  - CAS created mapping
    - Between tool-specific result types and test case CWEs
    - Tool results with a type mapped to the test case are “Positives”
  - Function name
    - “bad” → True Positive
    - “good” → False Positive
- Test cases with no True Positives have a False Negative added

Weakness Classes

Weakness Classes

- Results are analyzed by assigning each test case to one of 13 weakness classes
- Weakness classes are defined as a set of test case CWEs

Weakness Classes – 2010

Weakness Class	Example Weakness (CWE)	C/C++ Test Cases	Java Test Cases
Authentication and Access Control	CWE-620: Unverified Password Change	604	422
Buffer Handling	CWE-121: Stack-based Buffer Overflow	11,386	-
Code Quality	CWE-561: Dead Code	440	410
Control Flow Management	CWE-362: Race Condition	579	509
Encryption and Randomness	CWE-328: Reversible One-Way Hash	298	950
Error Handling	CWE-252: Unchecked Return Value	2,790	437
File Handling	CWE-23: Relative Path Traversal	2,520	718
Information Leaks	CWE-534: Information Leak Through Debug Log Files	283	468
Initialization and Shutdown	CWE-415: Double Free	9,894	450
Injection	CWE-89: SQL Injection	6,882	5,970
Miscellaneous	CWE-480: Use of Incorrect Operator	2,304	222
Number Handling	CWE-369: Divide by Zero	6,017	2,802
Pointer and Reference Handling	CWE-476: NULL Pointer Dereference	1,308	425

Precision, Recall, and F-Score

Justification

- CAS is concerned with two things:
  - What flaws does the tool report?
  - What non-flaws does the tool incorrectly report as a flaw? (false positives)
- CAS uses concepts from Information Retrieval in examination of static analysis tool results
  - Precision
  - Recall
  - F-Score

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”

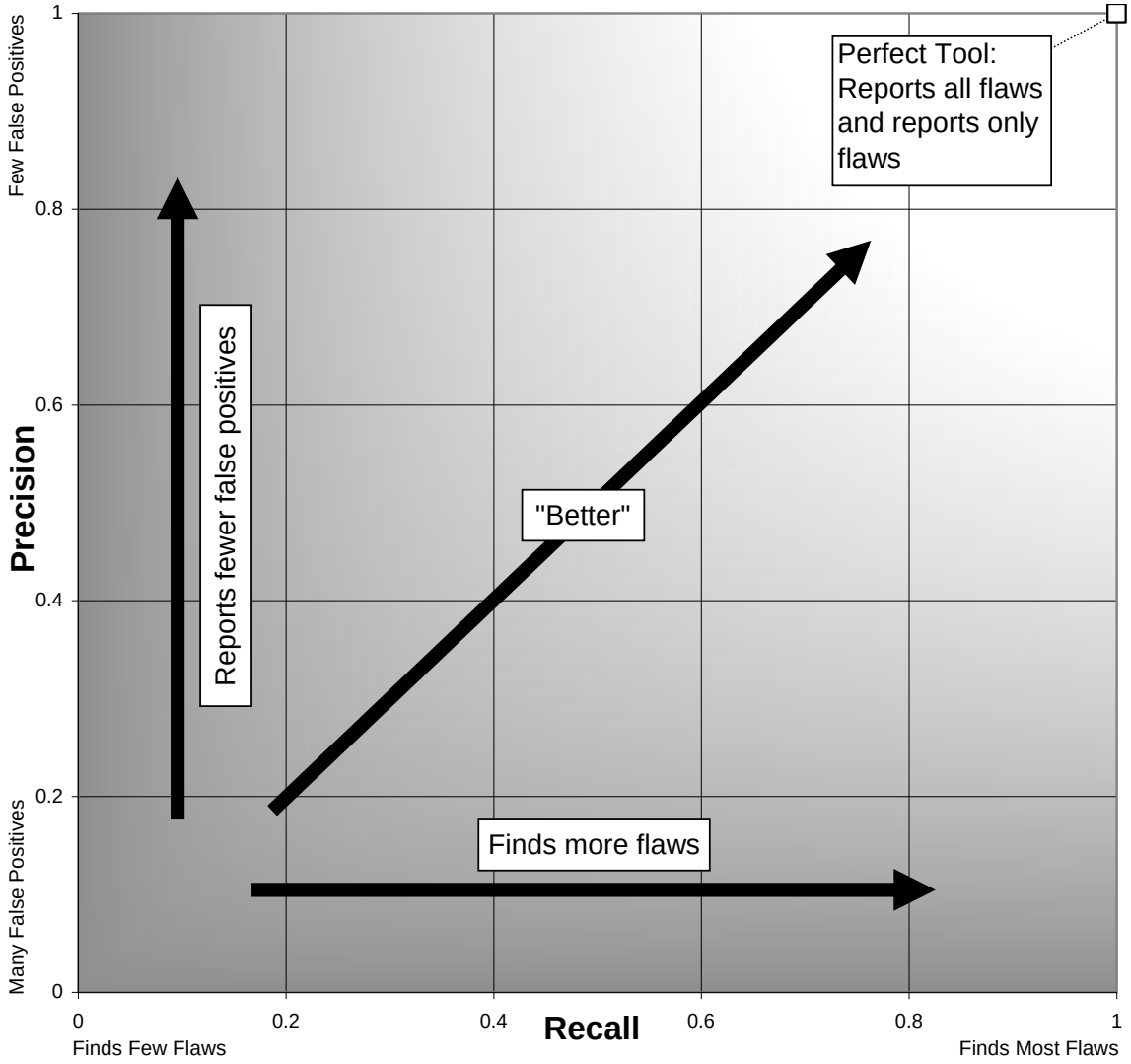
F-Score

- F-Score is defined as the harmonic mean of Precision and Recall

$$F\text{-}Score = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall} \right)$$

- Combines values into one value to compare
- Tends toward lower value
- Less than arithmetic mean (unless Precision and Recall are equal)

Precision-Recall Graph



Discriminations

Justification

- Precision, Recall, and F-Score on test cases don't tell whole story
- Unsophisticated "grep-like" tool can get:
  - Recall: 1
  - Precision: 0.5
  - F-Score: 0.67
  - Doesn't accurately reflect that tool is noisy
- Limitation of CAS test cases
  - Only 1 or 2 non-flaws for each flaw

Discrimination

- A “Discrimination” is a test case where a tool:
  - Correctly reported the flaw
  - Did not report any false positives
    - That is, did not erroneously report any flaws in locations where no flaw exists
- Each tool gets 0 or 1 discrimination(s) for each test case

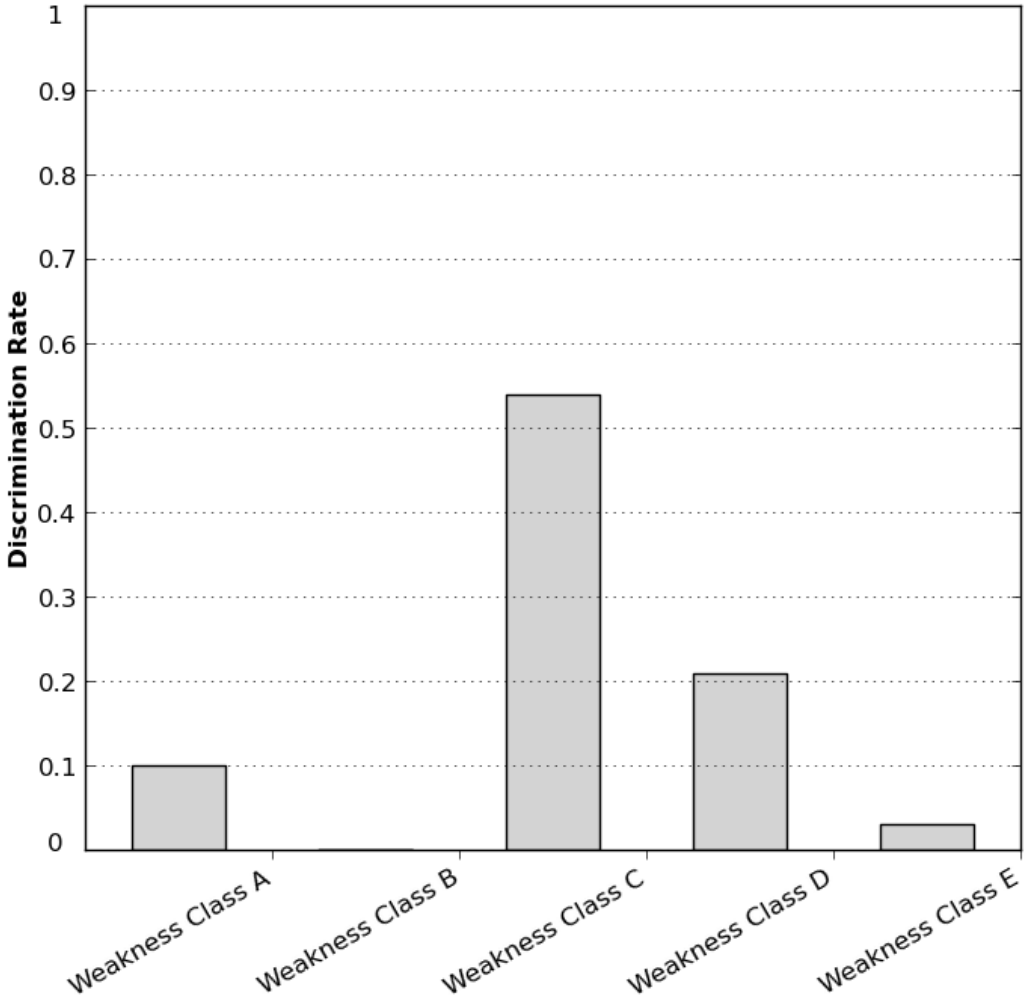
Discrimination Rate

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

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

- Discrimination Rate ≤ Recall
  - Every Discrimination "counts" toward Discrimination Rate and Recall
  - Every True Positive "counts" toward Recall, but not necessarily toward Discrimination Rate

Example Disc. Rate Graph



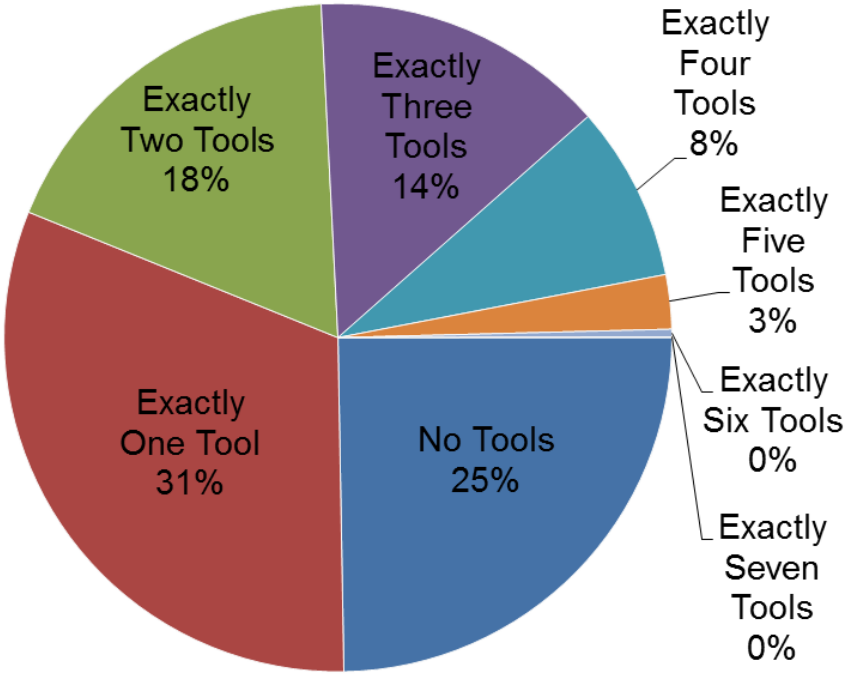
2010 Study Conclusions

2010 Study Conclusions

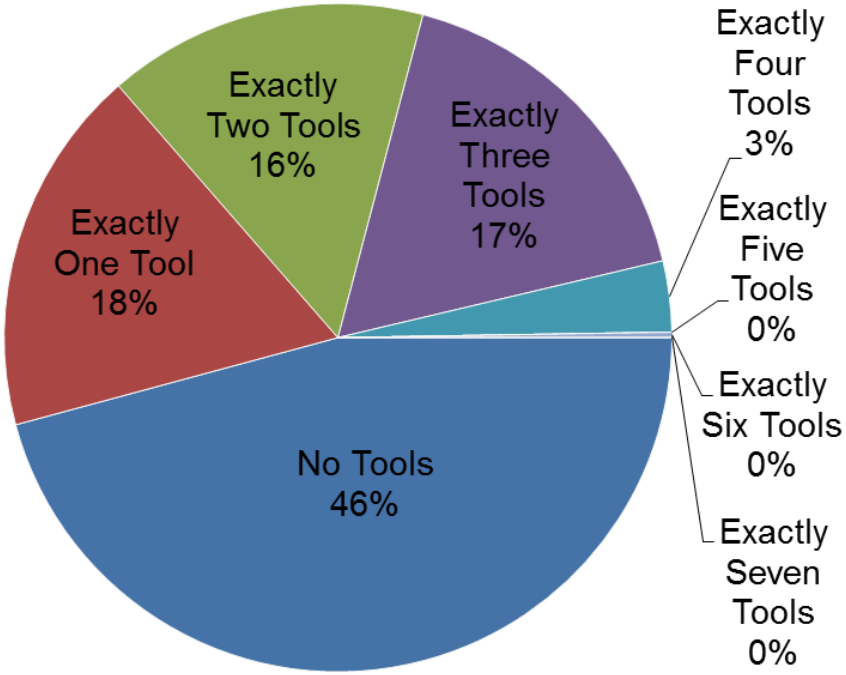
- Tools are not interchangeable
- Tools perform differently on different languages
- Complementary tools can be combined to achieve better results
- Each tool failed to report a significant portion of the flaws studied
  - Average tool covered 8 of 13 Weakness Classes
  - Average tool covered 22% of flaws in Weakness Classes covered

Flaws Reported – 2010

C/C++ Test Cases (2010)

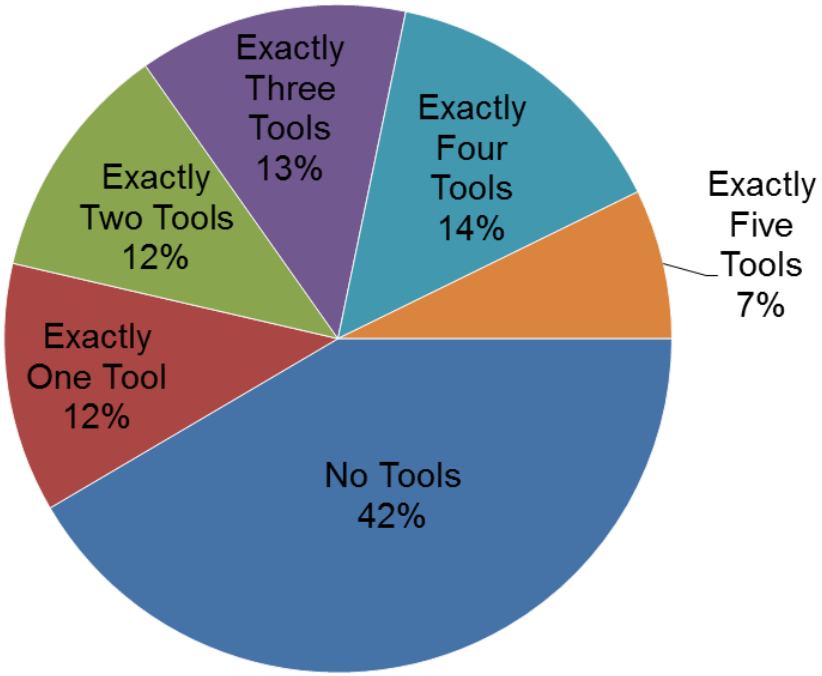


Java Test Cases (2010)



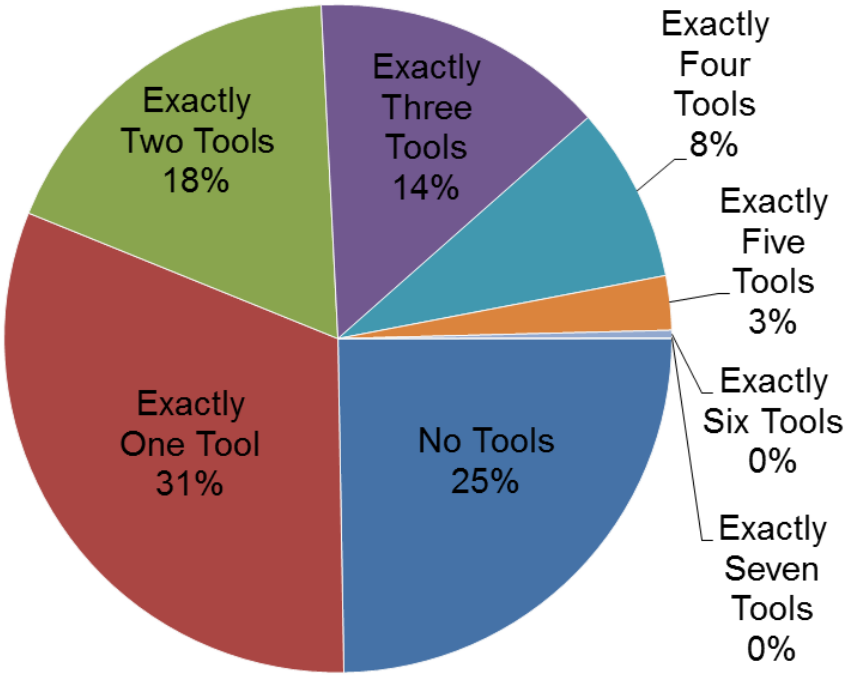
Flaws Reported – C/C++ 2009 vs. 2010

C/C++ Test Cases (2009)



- **Five tools**
- **207 Test Cases**
- **207 flaw types**
- **No data or control flows**

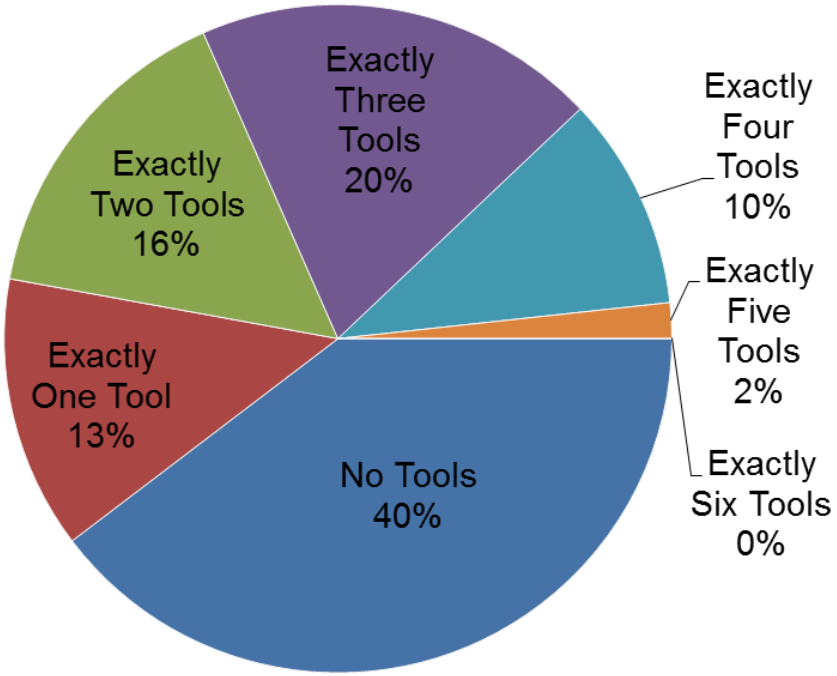
C/C++ Test Cases (2010)



- **Seven tools**
- **45,286 Test Cases**
- **1,432 flaw types**
- **Various data and control flows**

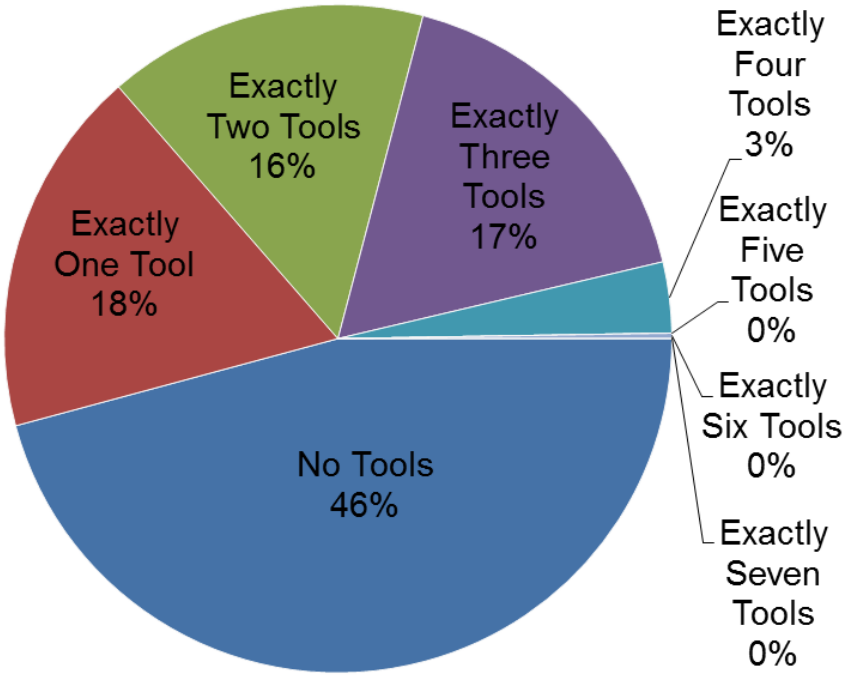
Flaws Reported – Java 2009 vs. 2010

Java Test Cases (2009)



- **Six tools**
- **174 Test Cases**
- **174 flaw types**
- **No data or control flows**

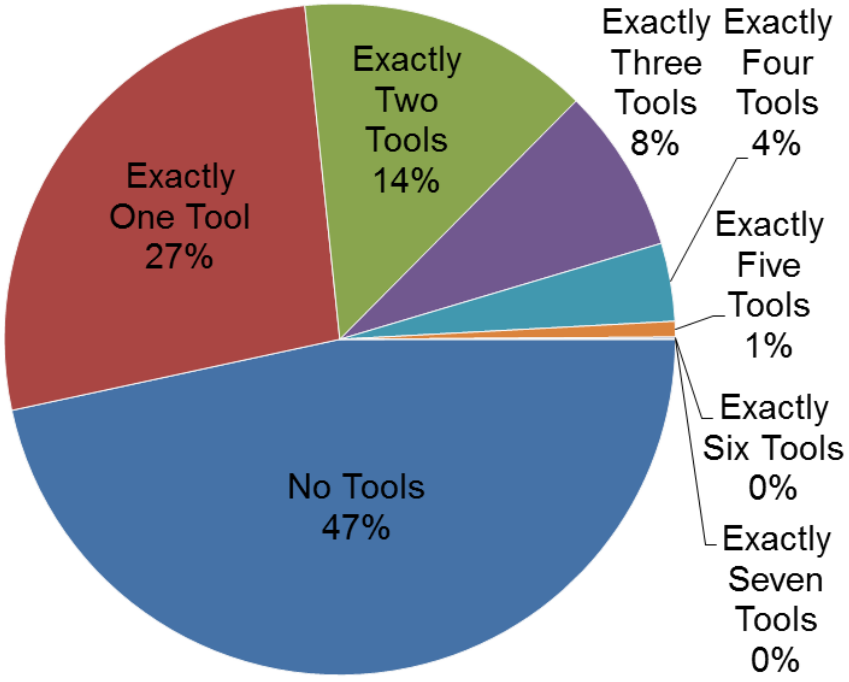
Java Test Cases (2010)



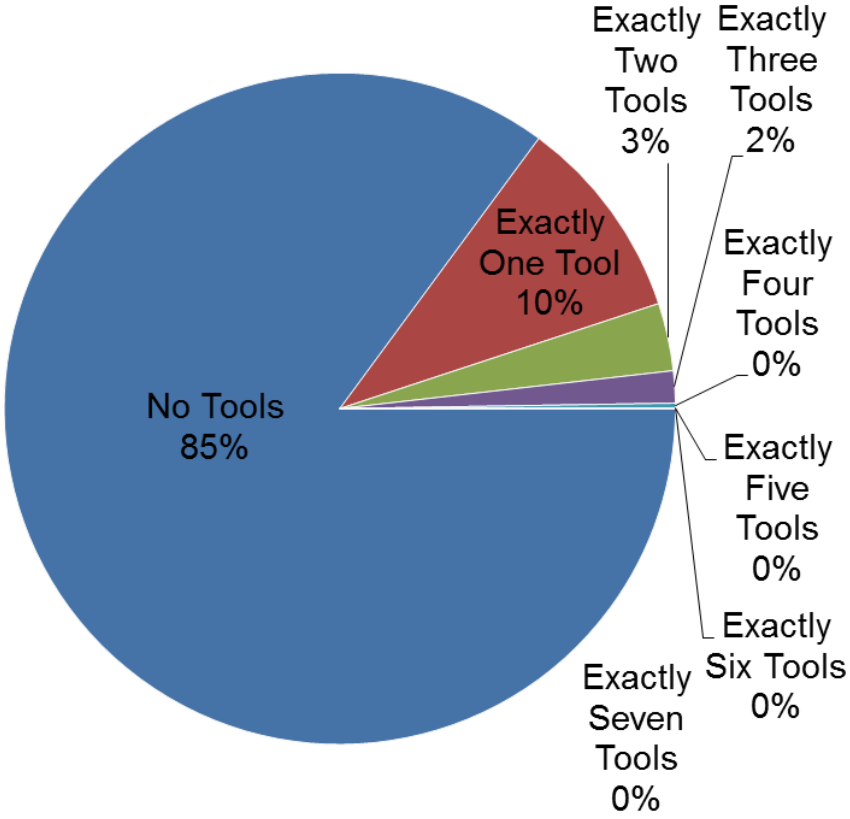
- **Seven tools**
- **13,801 Test Cases**
- **527 flaw types**
- **Various data and control flows**

Flaws Discriminated – 2010

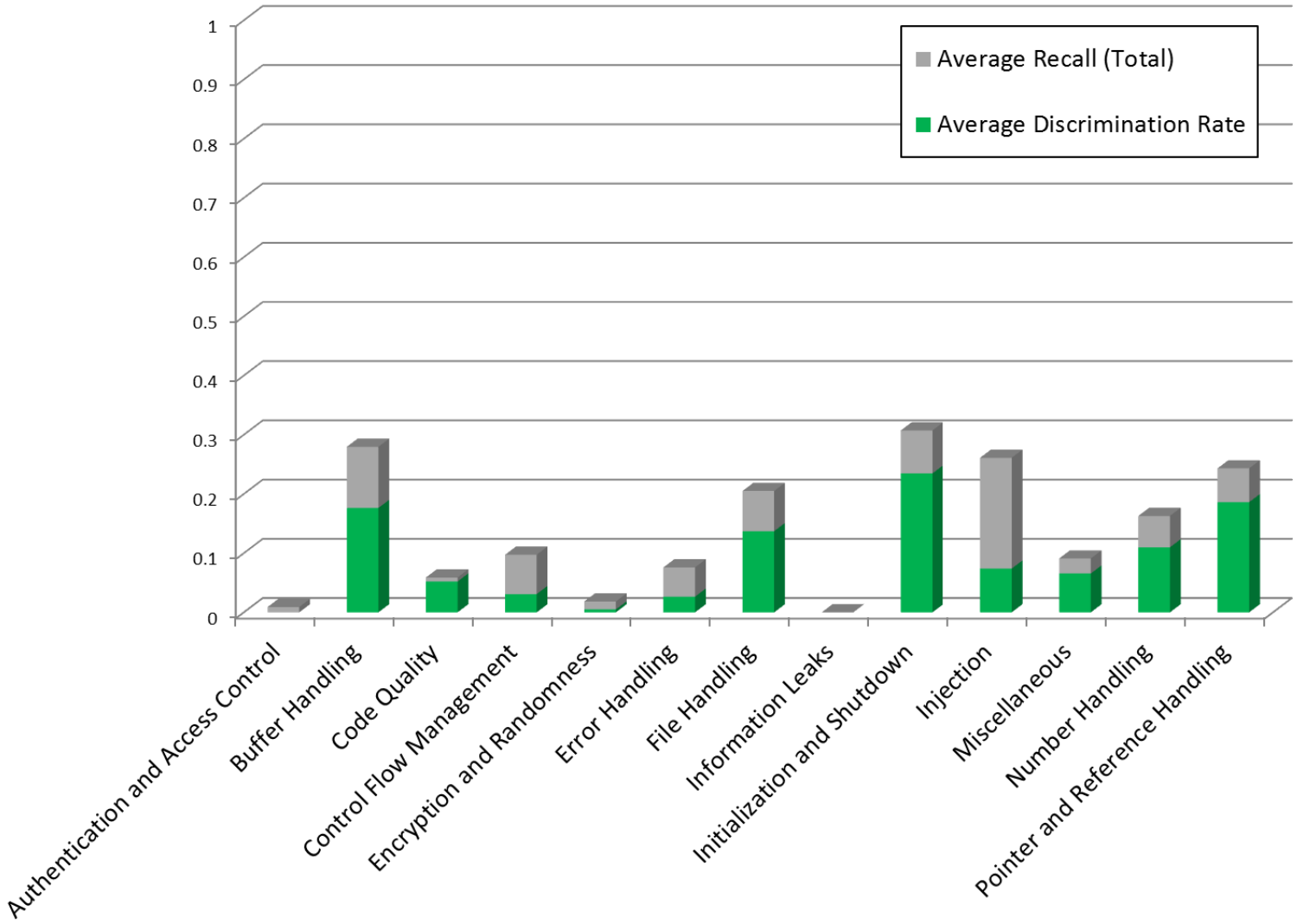
C/C++ Test Cases (2010)



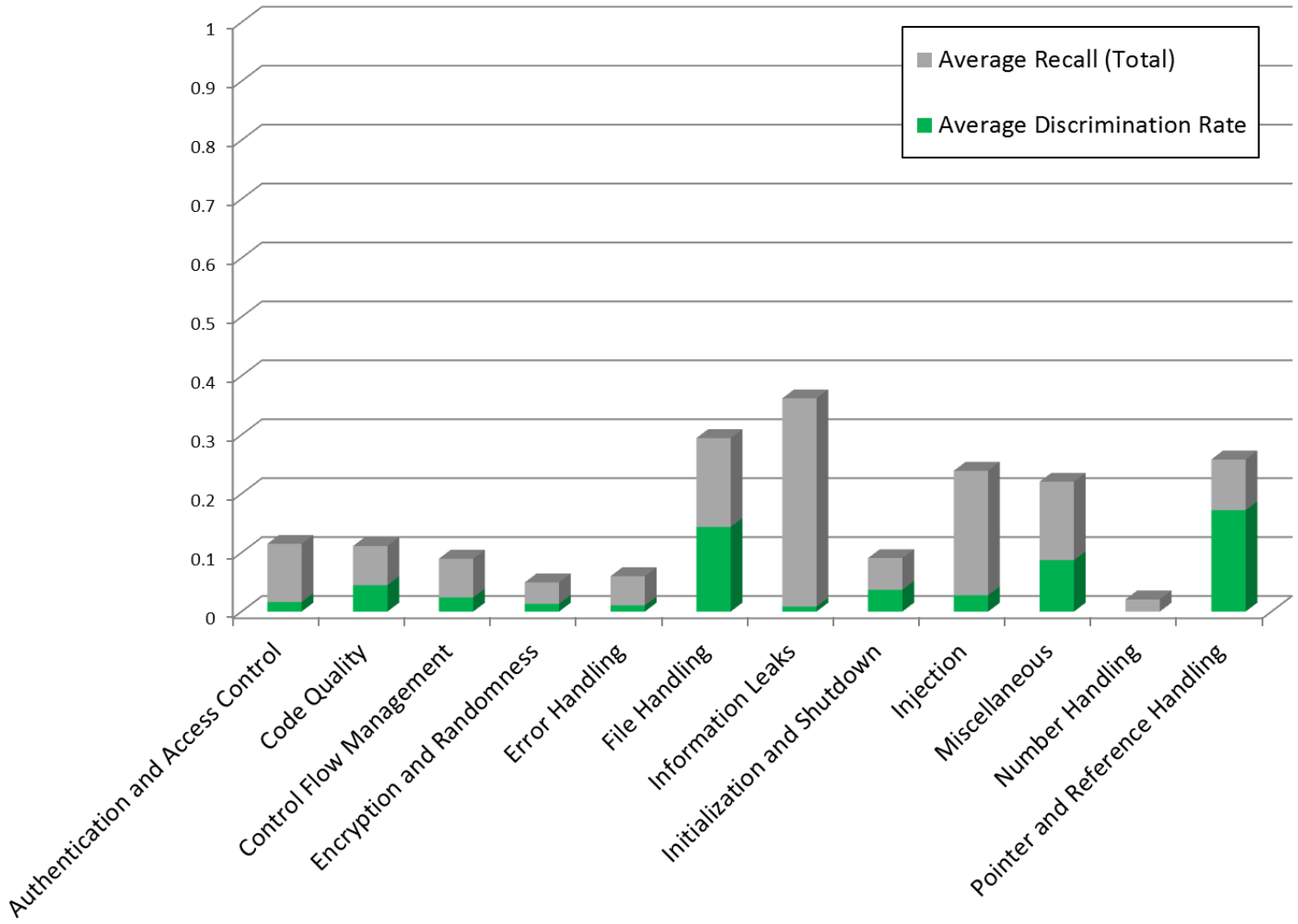
Java Test Cases (2010)



Flaws Reported and Disc. – C/C++ – 2010



Flaws Reported and Disc. – Java – 2010



Open Source vs. Commercial Tools

- Open source C/C++ tool was limited overall
  - Reported the flaws in a below-average fraction of the test cases in every Weakness Class it covered
  - Reported an above-average number of False Positives on five of the seven Weakness Classes it covered

Open Source vs. Commercial Tools

- Two open source Java tools studied had mixed results on the Weakness Classes they covered
  - In three Weakness Classes, an open source tool was the strongest of all tools (based on F-Score)
    - Control Flow Management
    - Code Quality
    - Error Handling
  - In four Weakness Classes, at least one open source tool was stronger than at least one commercial tool
    - Information Leaks
    - Initialization and Shutdown
    - Injection
    - Miscellaneous
  - In two Weakness Classes, the open source tools were the weakest tools
    - Auth. and Access Control
    - Pointer and Reference Handling

2011 Study Plans

Study Plans for 2011

- Update and expand Test Cases based on community feedback
- Soliciting input from vendors on configuration settings to use with their tools
- Considering additional tools
- Study scheduled to start in October 2011

Sticking to the Facts

Scientific Study of Static Analysis Tools

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