Software Assurance: Crippling Coming Cyberassaults

Dr. Paul E. Black

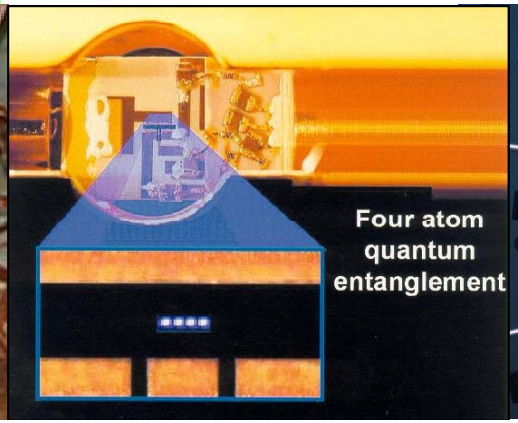
National Institute of Standards and Technology

http://samate.nist.gov/

paul.black@nist.gov

What is NIST?

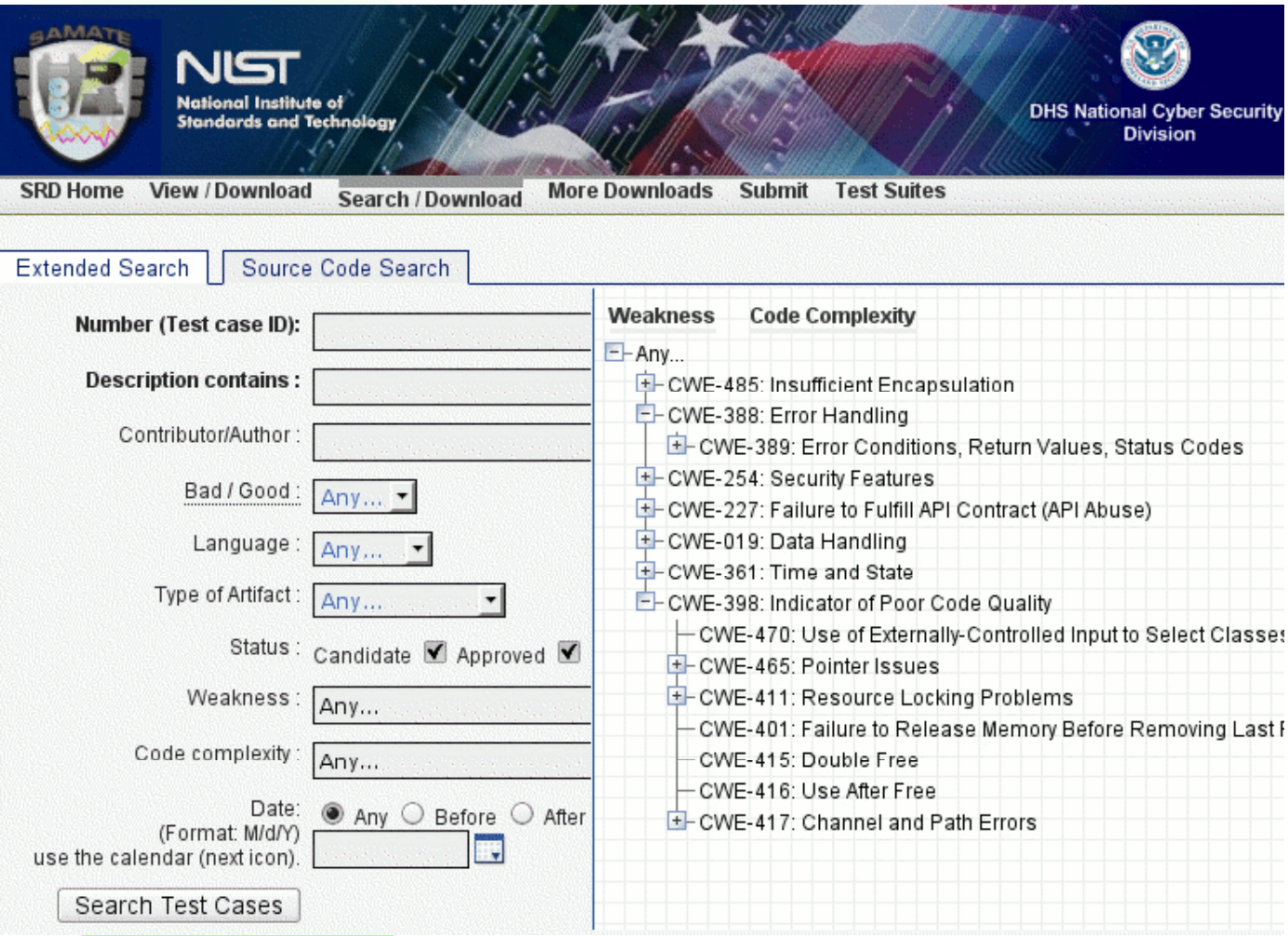
- **U.S. National Institute of Standards and Technology**
- **A non-regulatory agency in Dept. of Commerce**
- **3,000 employees + adjuncts**
- **Gaithersburg, Maryland and Boulder, Colorado**
- **Primarily research, not funding**
- **Over 100 years in standards and measurements: from dental ceramics to microspheres, from quantum computers to fire codes, from body armor to DNA forensics, from biometrics to text retrieval.**



The NIST SAMATE Project

- **Software Assurance Metrics And Tool Evaluation (SAMATE) project is sponsored in part by DHS**
- **Current areas of concentration**
  - **Web application scanners**
  - **Source code security analyzers**
  - **Static Analyzer Tool Exposition (SATE)**
  - **Software Reference Dataset**
  - ***Software labels***
  - ***Malware research protocols***
- **Web site http://samate.nist.gov/**

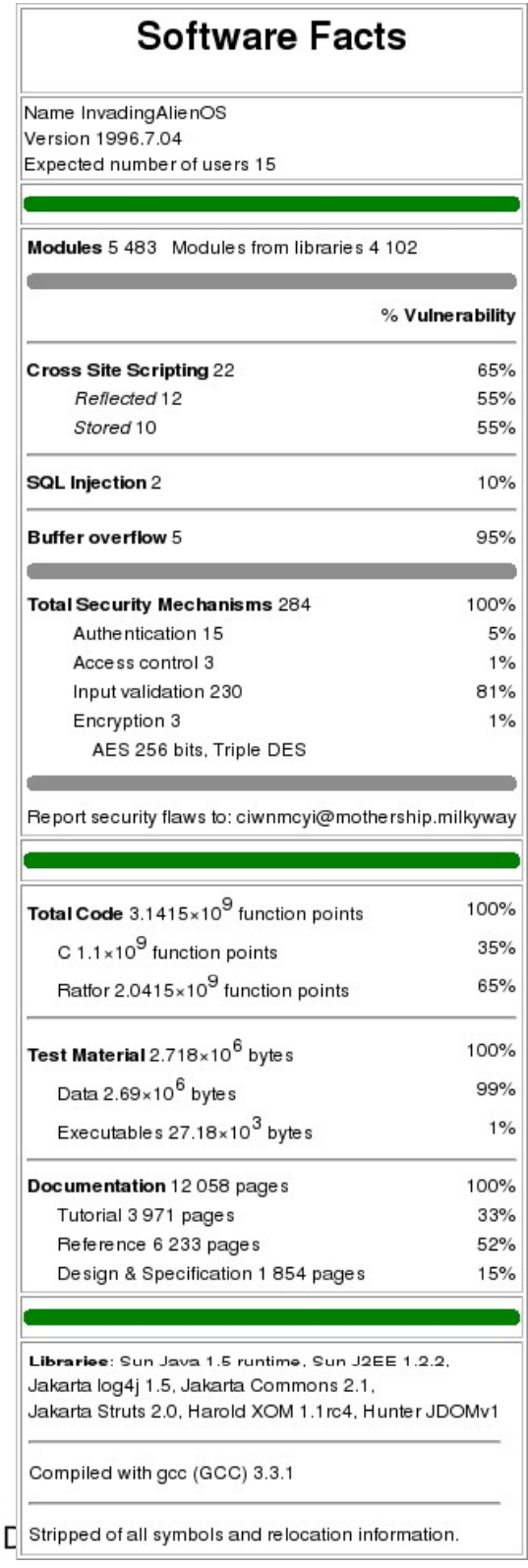
Software Reference Dataset



- **Public repository for software test cases**
- **Almost 1800 cases in C, C++, Java, and Python**
- **Search and compose custom Test Suites**
- **Contributions from Fortify, Defence R&D Canada, Klocwork, MIT Lincoln Laboratory, Praxis, Secure Software, etc.**

Software Facts Label

- **Software Facts should:**
  - **Voluntary**
  - **Absolutely simple to produce**
  - **Have a standard format for other claims**
- **What could be easily supplied?**
  - **Source available? Yes/No/Escrowed**
  - **Default installation is secure?**
  - **Accessed: network, disk, ...**
  - **What configuration files? (registry, ...)**
  - **Certificates (eg, "No Severe weaknesses found by CodeChecker ver. 3.2")**
- **Cautions**
  - **A label can give false confidence.**
  - **A label shut out better software.**
  - **Labeling diverts effort from real improvements.**



Software Facts

Name InvadingAlienOS
Version 1996.7.04
Expected number of users 15

Modules 5 483 Modules from libraries 4 102

	% **Vulnerability**
Cross Site Scripting 22	65%
Reflected 12	55%
Stored 10	55%
SQL Injection 2	10%
Buffer overflow 5	95%
Total Security Mechanisms 284	100%
Authentication 15	5%
Access control 3	1%
Input validation 230	81%
Encryption 3	1%
AES 256 bits, Triple DES	

Report security flaws to: ciwnmcyi@mothership.milkyway

Total Code 3.1415×10^{9} function points	100%
C 1.1×10^{9} function points	35%
Ratfor 2.0415×10^{9} function points	65%
Test Material 2.718×10^{6} bytes	100%
Data 2.69×10^{6} bytes	99%
Executables 27.18×10^{3} bytes	1%
Documentation 12 058 pages	100%
Tutorial 3 971 pages	33%
Reference 6 233 pages	52%
Design & Specification 1 854 pages	15%

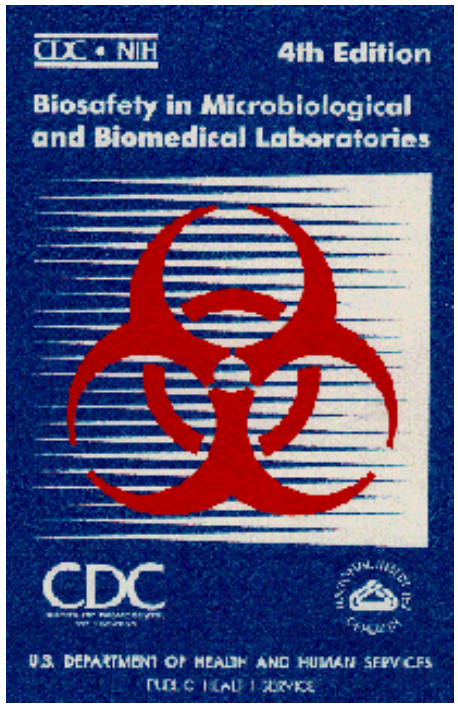
Libraries: Sun Java 1.5 runtime, Sun J2EE 1.2.2, Jakarta log4j 1.5, Jakarta Commons 2.1, Jakarta Struts 2.0, Harold XOM 1.1rc4, Hunter JDOMv1

Compiled with gcc (GCC) 3.3.1

Stripped of all symbols and relocation information.

Researching Risky Software

- **Many people research malware, but there are no widely accepted protocols.**
- **Biological research has defined levels with associated practices, safety equipment, and facilities.**
- **Some approaches are**
  - **Weakened programs (auxotrophs)**
  - **Programs that ALERT**
  - **Outgoing firewalls**
  - **Isolated networks**



- **Assurance that software is less vulnerable to coming cyberassaults**
- **Static and dynamic analysis**
- **Static Analysis Tool Exposition - 2009 outcomes and 2010 progress**

Assurance from three sources

$$A = f(p, s, e)$$

where *A* is functional assurance, *p* is process quality, *s* is assessed quality of software, and *e* is execution resilience.

p is process quality

- **High assurance software must be developed with care, for instance:**
  - **Validated requirements**
  - **Good system architecture**
  - **Security designed- and built in**
  - **Trained programmers**

s is assessed quality of software

- **Two general kinds of software assessment:**
  - **Static analysis**
    - **e.g. code reviews and scanner tools**
    - **examines code**
  - **Testing (dynamic analysis)**
    - **e.g. penetration testing, fuzzing, and red teams**
    - **runs code**

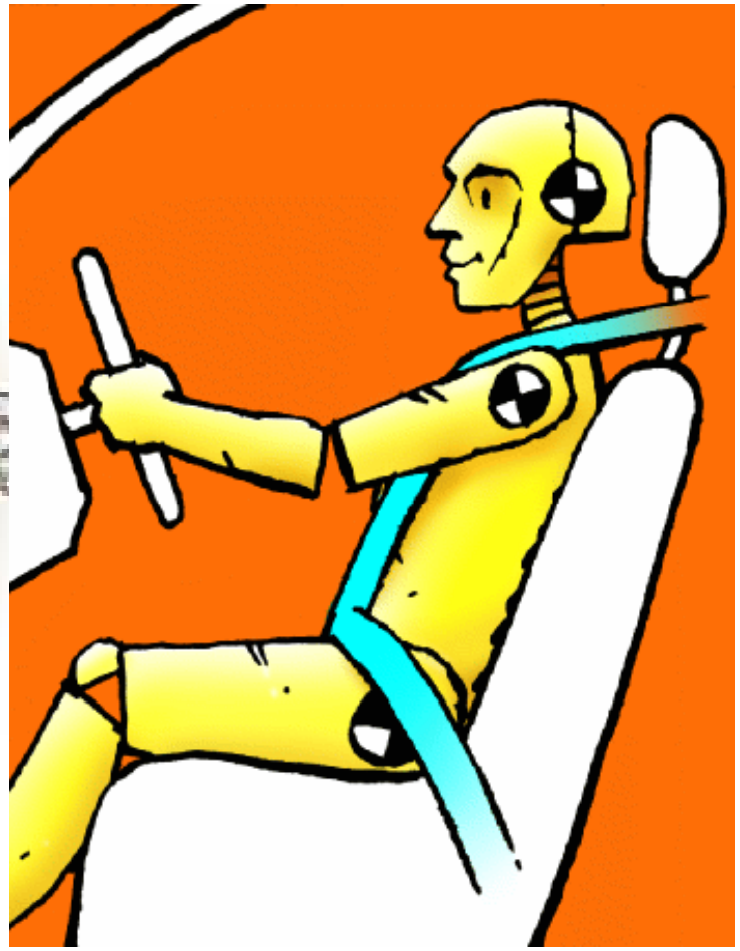
e is execution resilience

- **The execution platform can add assurance that the system will function as intended.**
- **Some techniques are:**
  - **Randomize memory allocation**
  - **Execute in a “sandbox” or virtual machine**
  - **Monitor execution and react to intrusions**
  - **Replicate processes and vote on output**

Software analysis is vital

- **Benefits are:**
  - **Provide feedback to development process**
  - **Build product assurance when process is less visible**
    - **contractors**
    - **open source**
    - **legacy software**
  - **Confirm minimum quality for execution**

Analysis is like a seatbelt …

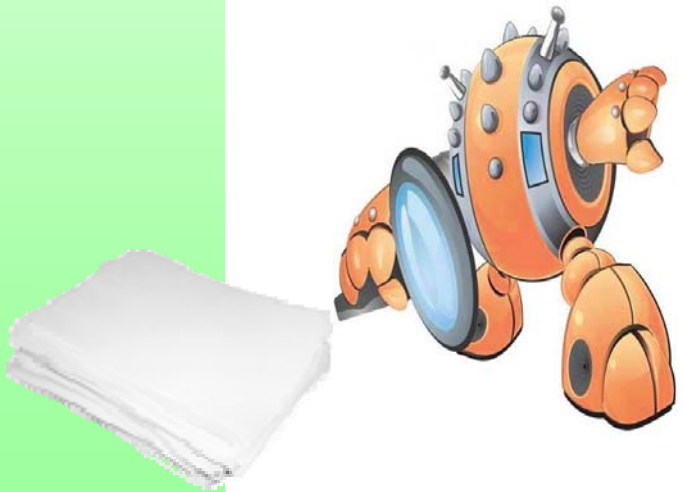


- Assurance that software is less vulnerable to coming cyberassaults
- **Static and dynamic analysis**
- Static Analysis Tool Exposition - 2009 outcomes and 2010 progress

Comparing Static Analysis with Dynamic Analysis

Static Analysis

- Code review
- Binary, byte, or source code scanners
- Model checkers & property proofs
- Assurance case



Dynamic Analysis

- Execute code
- Simulate design
- Fuzzing, coverage, MC/DC, use cases
- Penetration testing
- Field tests

Strengths of Static Analysis

- **Applies to many artifacts, not just code**
- **Independent of platform**
- **In theory, examines *all possible* executions, paths, states, etc.**
- **Can focus on a single specific property**

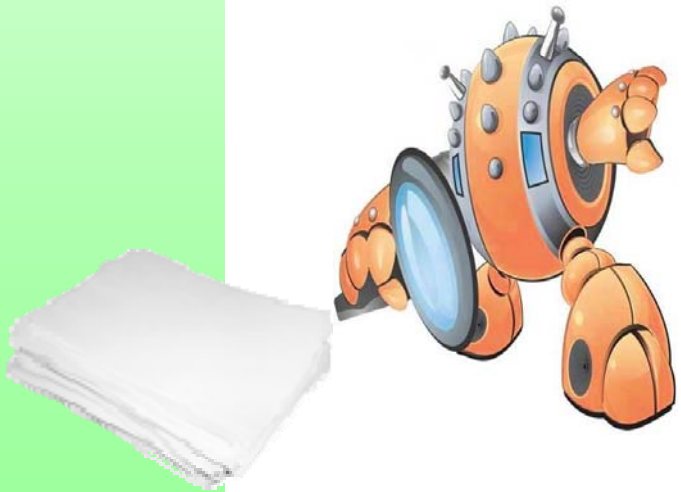
Strengths of Dynamic Analysis

- **No need for code**
- **Conceptually easier - “if you can run the system, you can run the test”.**
- **No (less) need to build or validate models or make assumptions.**
- **Checks installation and operation, along with end-to-end or whole-system.**

Static and Dynamic Analysis Complement Each Other

Static Analysis

- Handles unfinished code
- Can find backdoors, eg, full access for user name "JoshuaCaleb"
- Potentially complete



Dynamic Analysis

- Code not needed, eg, embedded systems
- Has few(er) assumptions
- Covers end-to-end or system tests
- Assess as-installed

- Assurance that software is less vulnerable to coming cyberassaults
- Static and dynamic analysis
- **Static Analysis Tool Exposition - 2009 outcomes and 2010 progress**

Static Analysis Tool Exposition (SATE) Overview

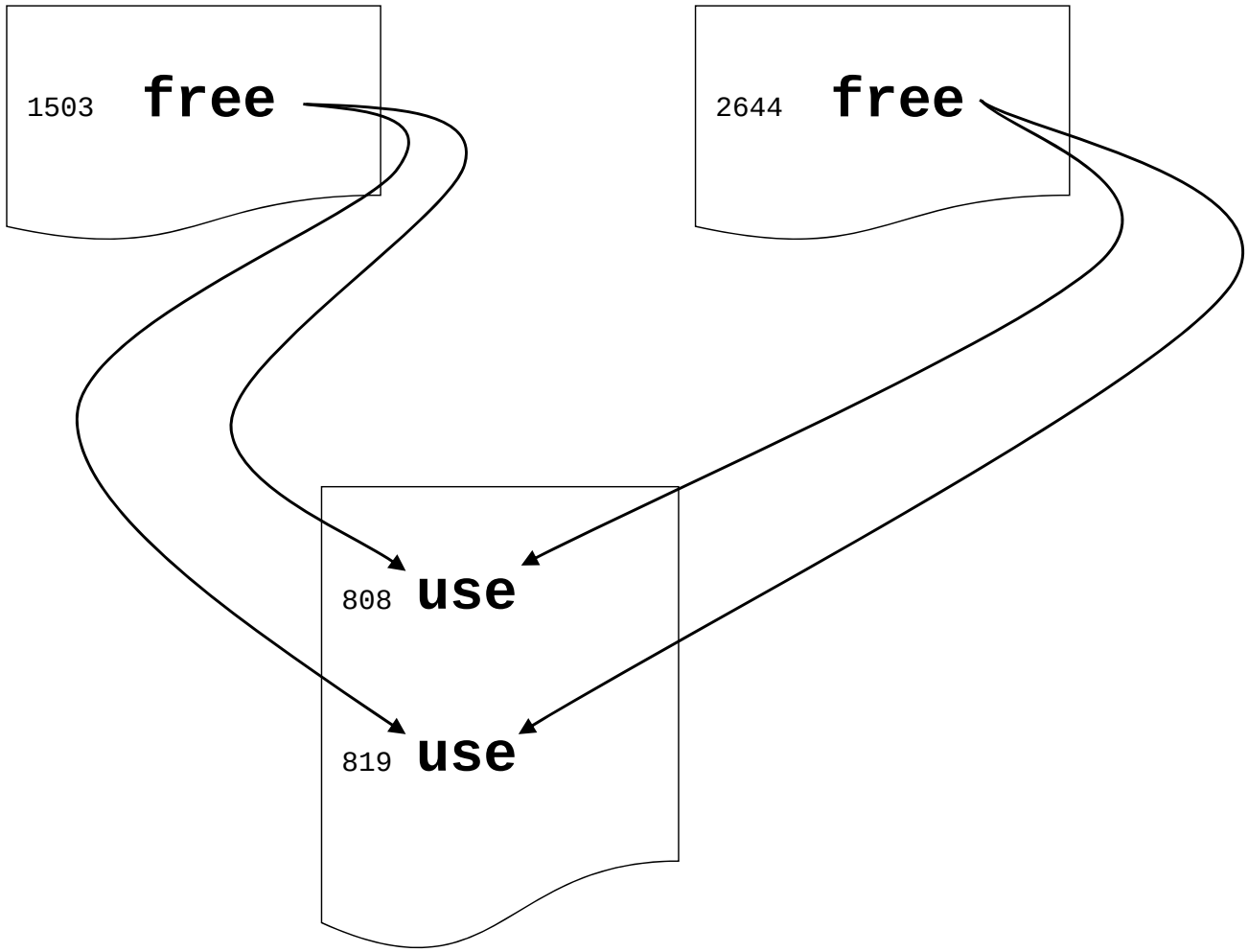
- **Goal: advance research in, and improvement of, static analysis tools for security-relevant defects and speed tool adoption by demonstrating use on real software.**
- **Checkpoints**
  - **Participants run tools on Java and C programs we choose**
  - **NIST-led researchers analyze reports**
  - **Everyone shares results and observations at a workshop**
  - **Later release final report and all data**
- **http://samate.nist.gov/SATE.html**
- **Co-funded by NIST and DHS/NCSD**

SATE Participants

- **2008:**
  - **Aspect Security ASC**
  - **Checkmarx CxSuite**
  - **Flawfinder**
  - **Fortify SCA**
  - **Grammatech CodeSonar**
  - **HP DevInspect**
  - **SofCheck Inspector for Java**
  - **UMD FindBugs**
  - **Veracode SecurityReview**
- **2009:**
  - **Armorize CodeSecure**
  - **Checkmarx CxSuite**
  - **Coverity Prevent**
  - **Grammatech CodeSonar**
  - **Klocwork Insight**
  - **LDRA Testbed**
  - **SofCheck Inspector for Java**
  - **Veracode SecurityReview**

“Number of bugs” is undefined

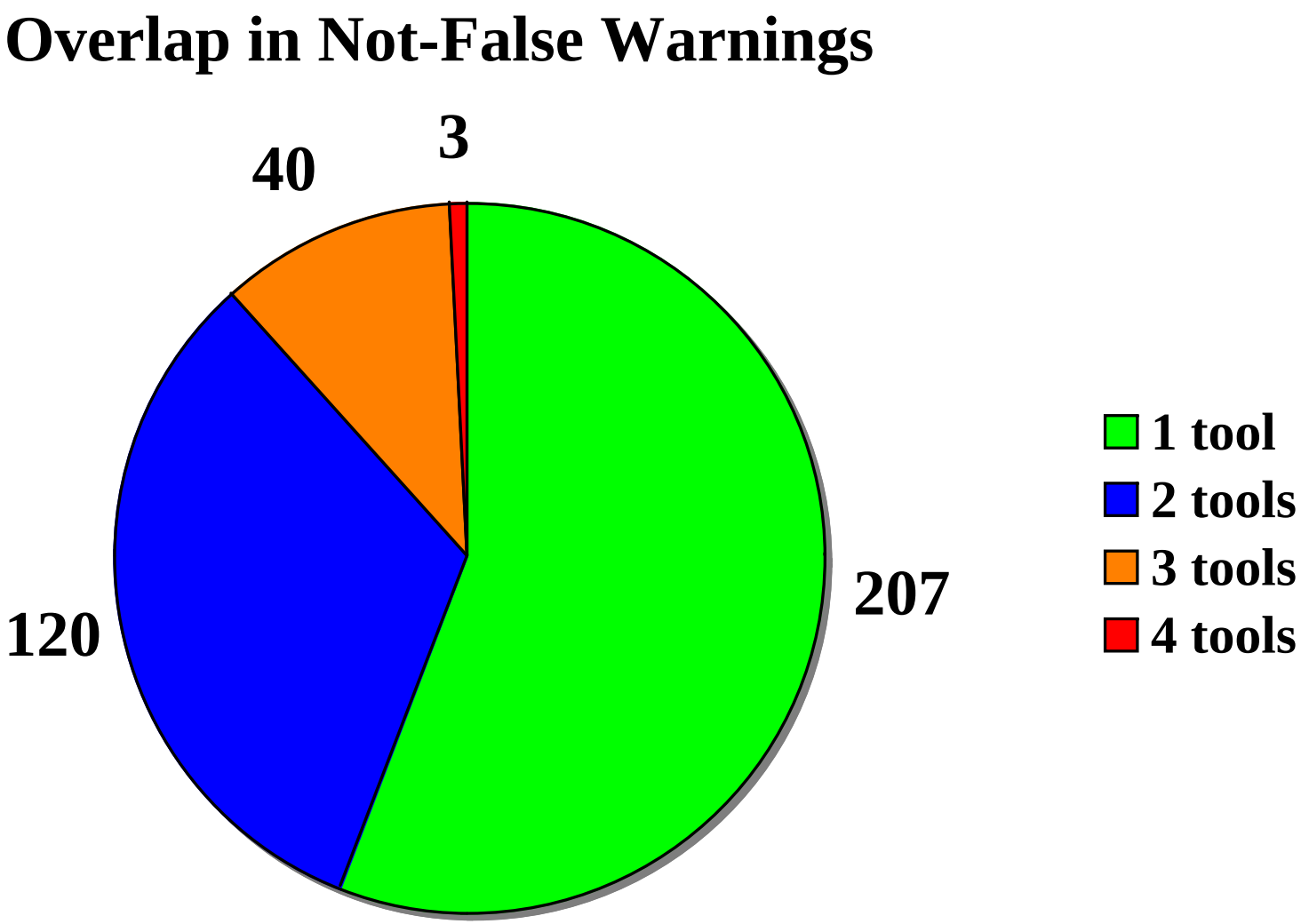
Tangled Flow: 2 sources, 2 sinks, 4 paths



Summary of 2009 tool reports

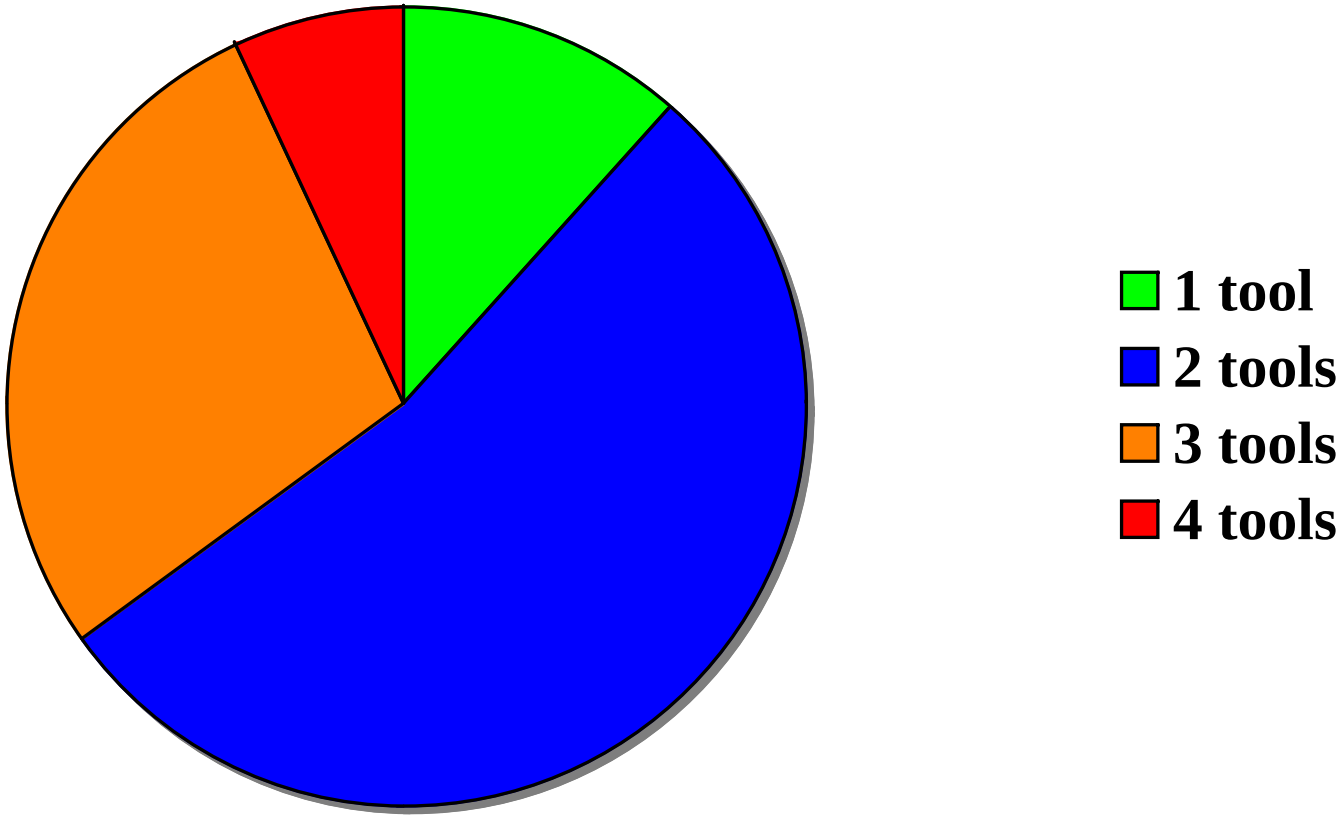
- **Reports from 18 tool runs**
- **About 20,000 total warnings**
  - **but tools prioritize by severity, likelihood**
- **Reviewed 521 warnings - 370 were not false**
- **Number of warnings varies a lot by tool and case**
- **83 CWE ids/221 weakness names**

Tools don’t report same warnings



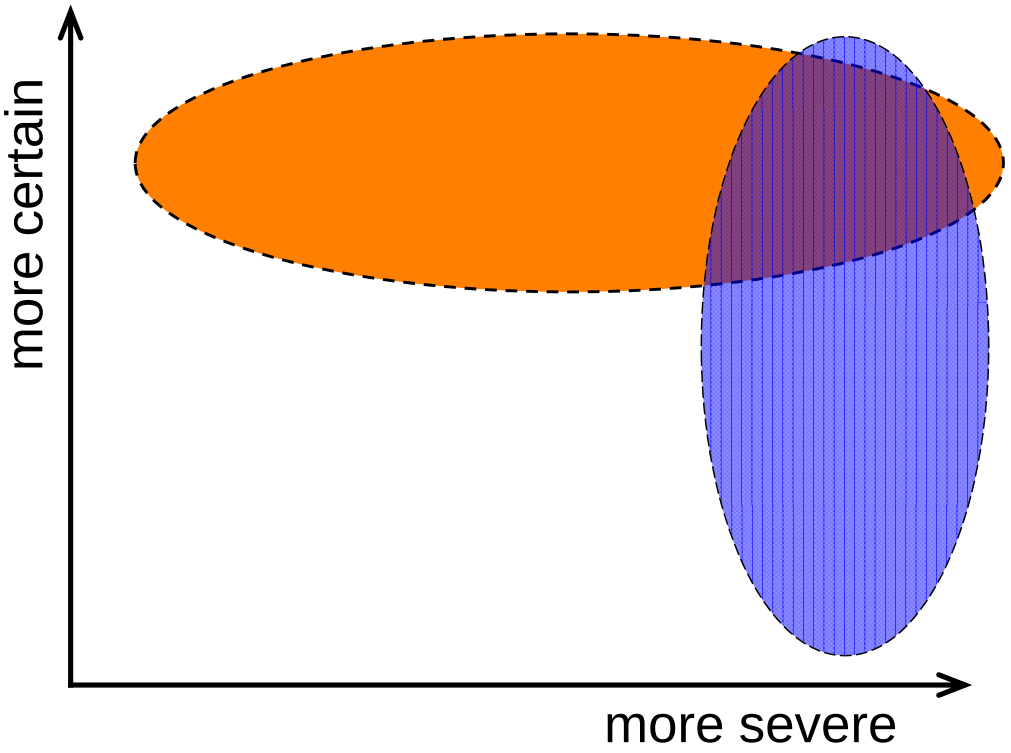
Some types have more overlap

Overlap in Not-False Buffer Errors

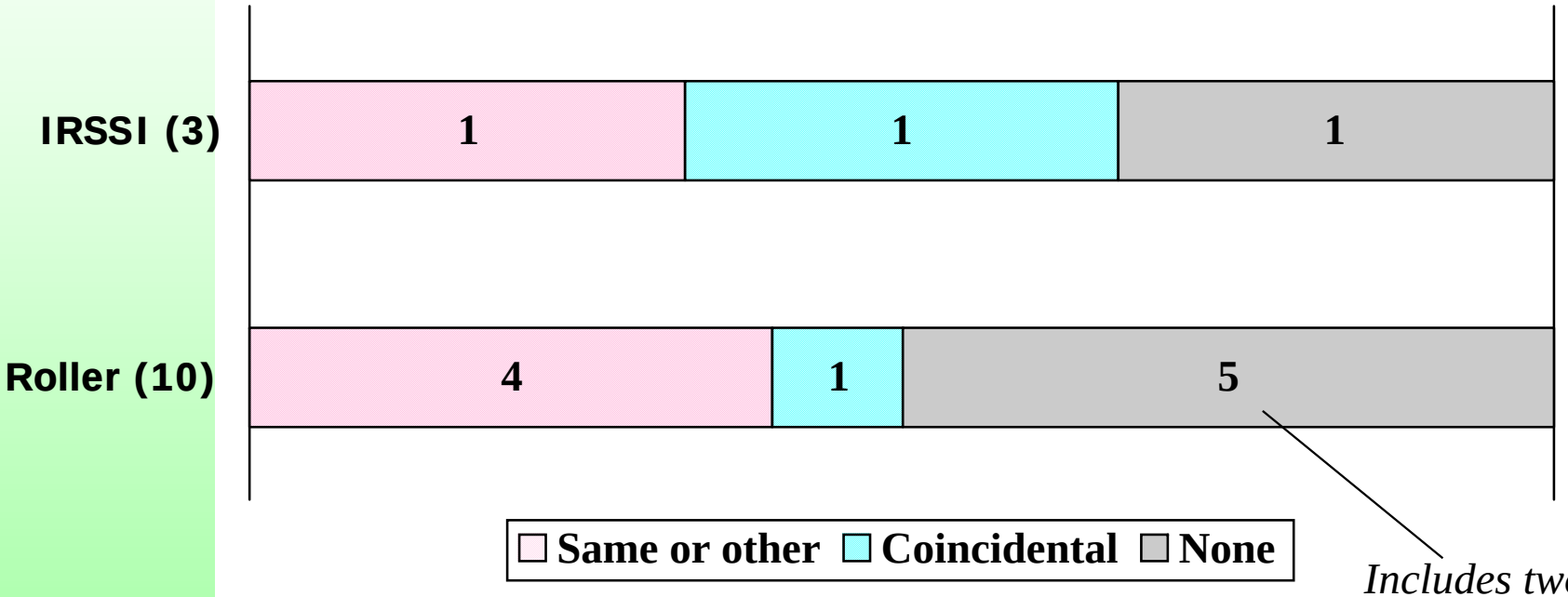


Why don’t tools find same things?

- **Tools look for different weakness classes**
- **Tools are optimized differently**



Tools find things that people find



SATE 2010 tentative timeline

- ✓ **Hold organizing workshop (12 Mar 2010)**
- ✓ **Recruit planning committee.**
- **Revise protocol.**
- **Choose test sets. Provide them to participants (17 May)**
- **Participants run their tools. Return reports (25 June)**
- **Analyze tool reports (27 Aug)**
- **Share results at workshop (October)**
- **Publish data (after Jan 2011)**

Acronyms

- **CWE - Common Weakness Enumeration**
  http://cwe.mitre.com/
- **DHS/NCSD - Department of Homeland Security/National Cyber Security Division**
- **MC/DC - Modified Condition/Decision Coverage**
- **SAMATE - Software Assurance Metrics And Tool Evaluation (project at NIST)**
- **SATE - Static Analysis Tool Exposition (annual event)**
- **NIST - National Institute of Standards and Technology**