SATE VI
Ockham Sound Analysis Criteria

Paul E. Black
paul.black@nist.gov

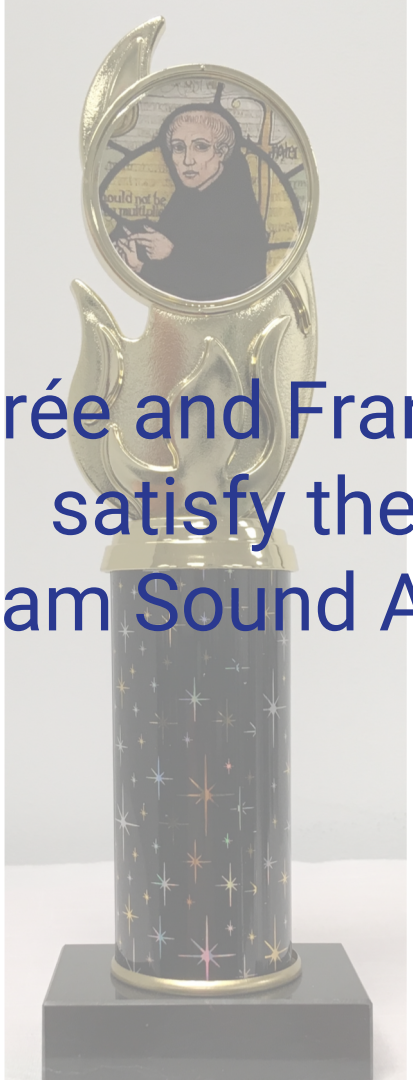
William of Ockham
from Wikipedia

NIST National Institute of Standards and Technology U.S. Department of Commerce

SAMATE

19 September 2019

Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor does it imply that the products are necessarily the best available for the purpose.



Astrée and Frama-C satisfy the SATE VI Ockham Sound Analysis Criteria

NIST National Institute of Standards and Technology U.S. Department of Commerce

SAMATE

SATE VI Ockham Sound Analysis Criteria

1. The tool is claimed to be sound.
2. The tool produces findings for at least 75 % of the sites.
3. Even one incorrect finding disqualifies a tool for this SATE.

https://samate.nist.gov/SATE6OckhamCriteria.html

NIST National Institute of Standards and Technology U.S. Department of Commerce

“Sound”—What Do We Mean?

- Precise; mathematically-based
- No heuristic analysis

- Since program analysis is undecidable, “I don’t know” or false positives are acceptable.

Wherefore Sound Analyzers & Ockham

- Advantage
  - Guarantees
- Disadvantages
  - Many weakness classes do not have a formal description
  - Takes more time to precisely match target environment
  - May require more computing resources

Juliet 1.3 C Test Suite

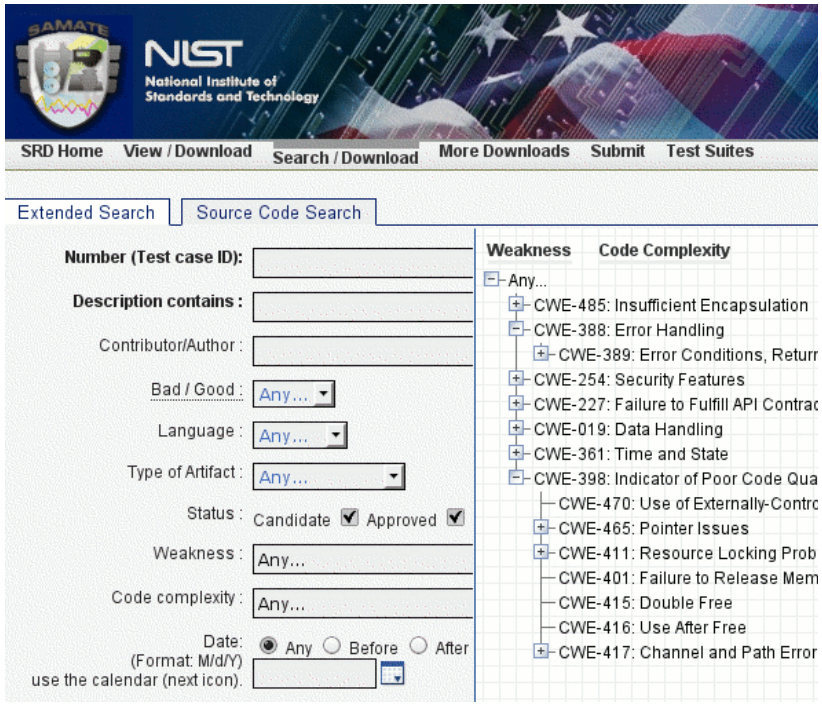
- Synthetic C programs
- Each test case is about 200 lines of code in one to six files
- Each test case has one “bad” function (with the bug) and one or more “good” functions (without the bug).
- 38 802 test cases under 118 weakness classes
- File of bug locations

- Original version developed by NSA’s Center for Assured Software (CAS) and released in 2010.

NIST National Institute of Standards and Technology U.S. Department of Commerce

SAMATE

SARD, by the way

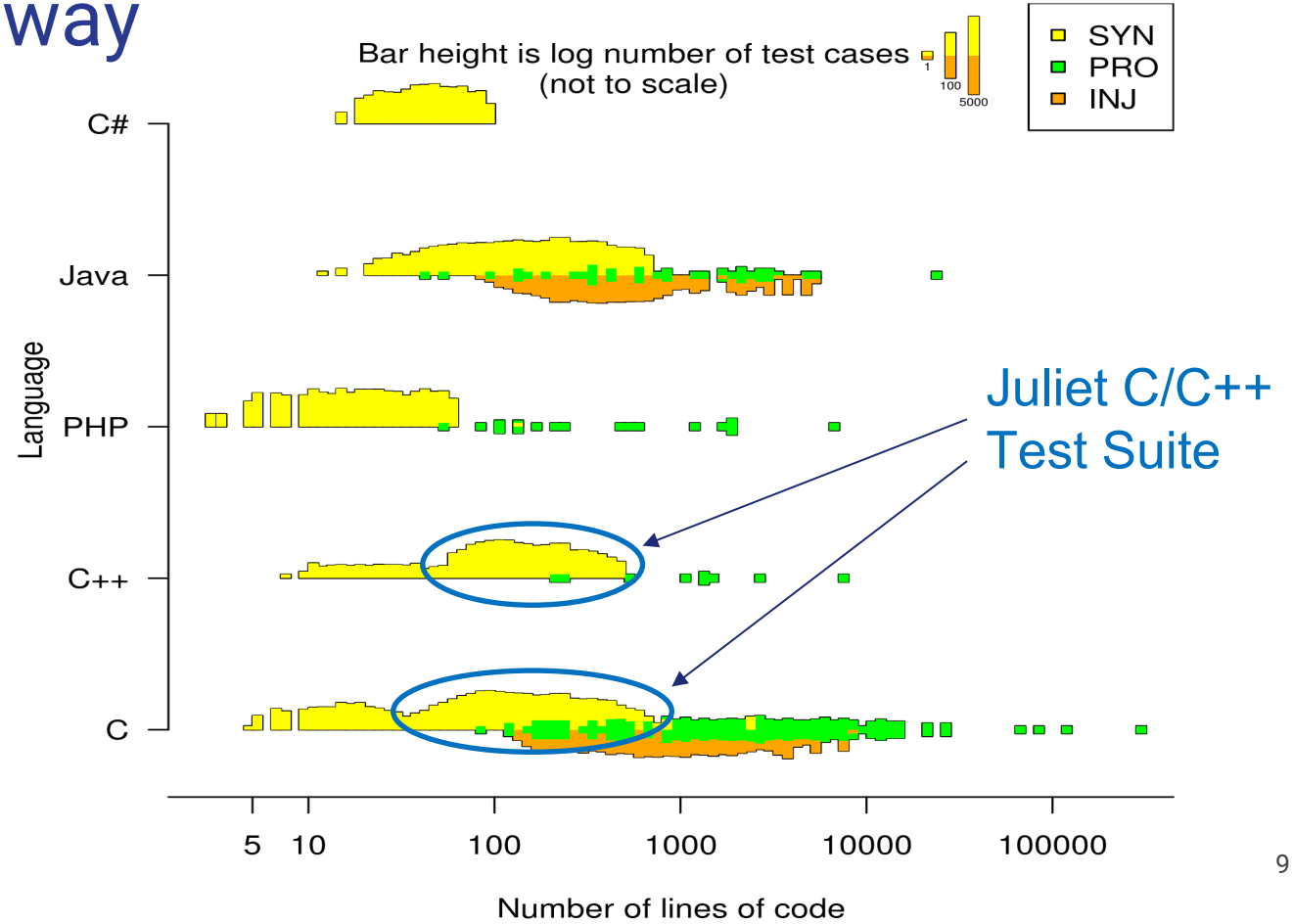


- **Public repository of software assurance test cases with known vulnerabilities**
- **Large, production programs to small, synthetic test cases**
- **Over 140 000 test cases in C, C++, Java, PHP, C#, and Python**
- **Contributions from IARPA, Fortify, TELECOM Nancy, Defence R&D Canada, Klocwork, MIT Lincoln Laboratory, Praxis, Toyota ITC, Secure Software, SATE, etc.**

- https://samate.nist.gov/SARD/

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

SARD, by the way



National Institute of Standards and Technology
U.S. Department of Commerce

Evaluation

- We ran the tools with invaluable help and guidance from tool makers

- Considered 19142 buggy sites from 40 Juliet CWE groups
- Classified them into 15 weakness classes
- Extracted 46651 tool warnings.

  (Tools produced far more, but this is all we extracted for Ockham.)

Results

- Found *thousands* of errors in the manifest of “known” bugs for Juliet 1.3 C.
- Astrée and Frama-C satisfy the SATE VI Ockham Sound Analysis Criteria.

What Did We Learn? 1

- Very precise analysis requires set up and tuning. Tools need a detailed description of the compile and execution environments.
  - For example, is an `int` 32 or 64 bits? Does the code rely on the compiler laying out memory for a `struct` in a certain order and without padding? Do you want warnings of unsigned short integer overflow, often used in hash or crypto computations? Is the high-order bit propagated when a signed integer is shifted right? How is floating-point addition rounded? The C11 standard allows for different behaviors of bitwise operators. The term “implementation-defined” occurs almost 200 times in the C11 standard.

What Did We Learn? 2

- Juliet is useful to calibrate tools.
  - For instance, if you want to catch bug class X, Juliet (probably) has test cases for it. These cases help you choose options, models, etc. and understand the output. You have confidence that the way you're running the tool reports class X if they exist.

- Automated scripts and programs were indispensable to rerun analysis when errors were found and to recheck former results.

- A common output format, like SARIF, would save us time.

What Did We Learn? 3

- Even with “well-defined” classes, it is still difficult to match and classify bugs
  - We tried to create logical, orthogonal, precise classes, following the Bugs Framework style.

Next Steps For Juliet

- Fix Juliet manifests
- Fix a dozen systematic problems in C source code
- Enrich SARD meta-information: add proof of vulnerability/exploit including expected outcome (no failure, anomalous behavior, crash, storage corruption, exploit), richer location information (source, sink, multiple lines), chains, etc.

Next Steps For Ockham Criteria

- In development: Juliet 2.0 (Kilo?) with cases in C#, JavaScript, and Python.
- Perhaps two additional tools for *this* SATE: we have one installed, but haven't had time to run it and analyze results.
- Focus on formal property prover, not (just) bug finding?
  - Formal Methods (FM) Rodeo
  - FM test cases, like SV-COMP and RERS
  - Definition of complexity characteristics – what makes analysis hard?