SAND2019-5232C

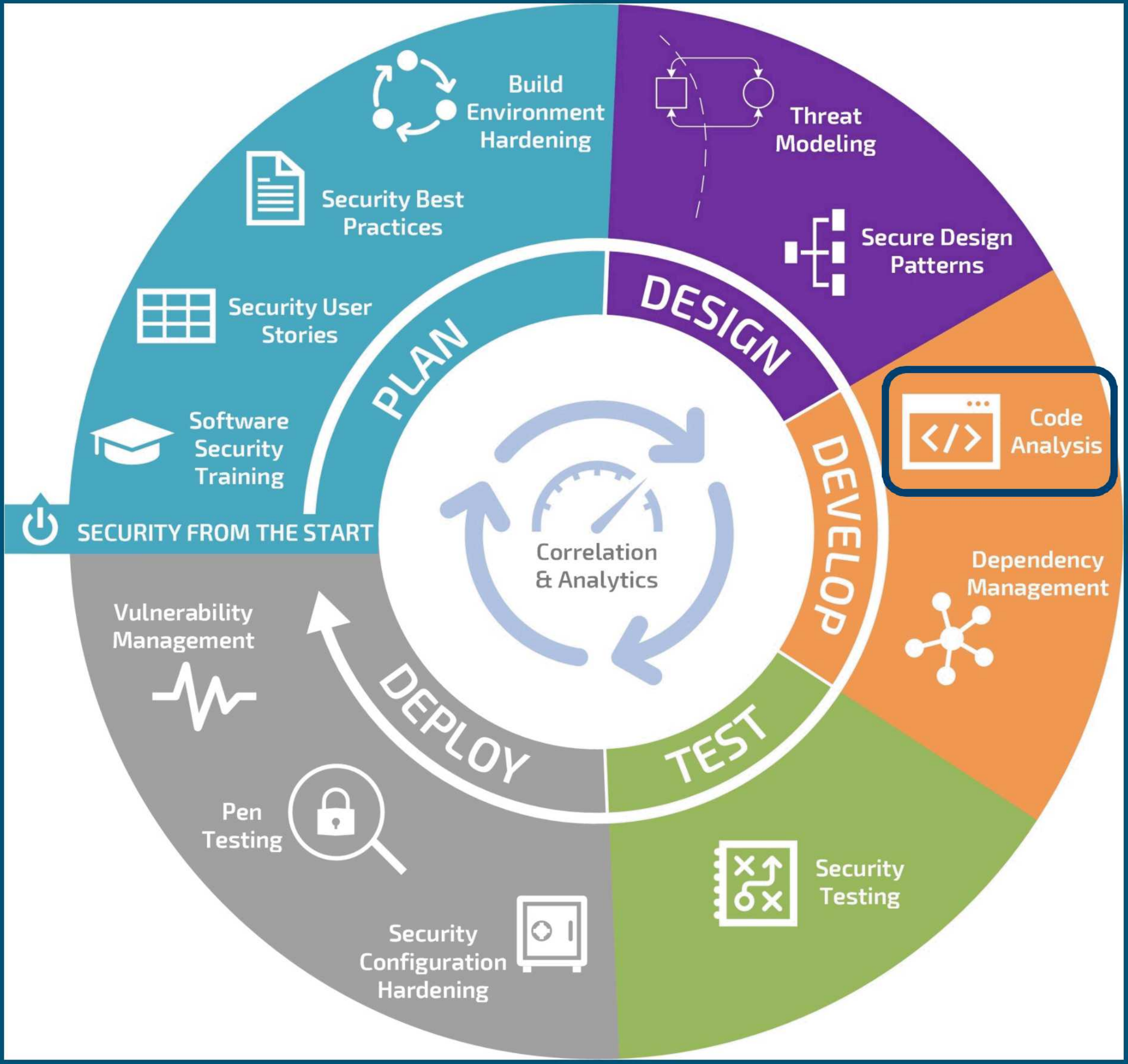
Choosing Static Application Security Testing Tools

PRESENTED BY

Roger Hartley [rthartl@sandia.gov]

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

The Secure Software Development Lifecycle



Overview

- Why SAST tool selection is a problem
- SAST tool Studies
- How they work (open-source)
- Metrics
- Classification
- Test suites: production vs synthetic
- (Highly) sampled results
- How to choose a SAST tool

Why SAST tool selection is a problem

Bottom line:

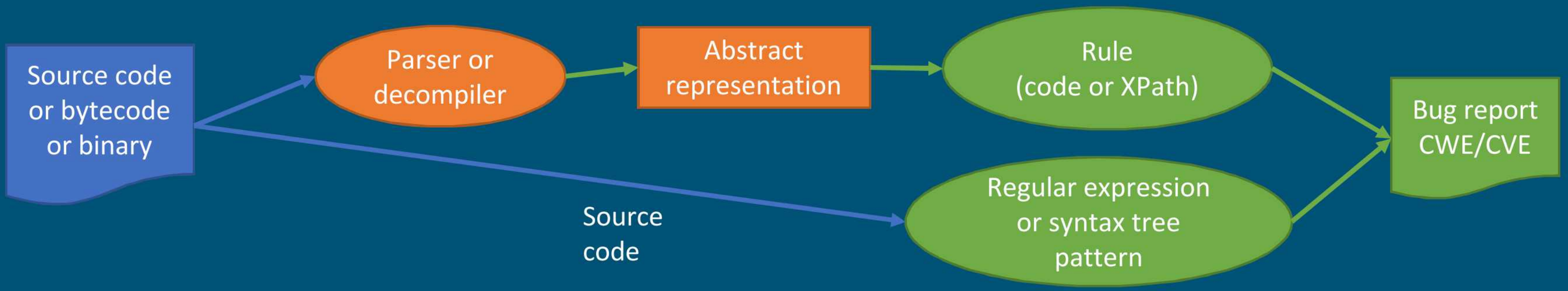
- ❖ No single SAST Tool will find all or even most flaws
- ❖ All SAST Tools have significant false positive rates
- ❖ To be of any use, each finding has to be examined whether it is True Positive (TP) or False Positive (FP)
- ❖ Tools are not interchangeable; they have different coverage of flaws
- ❖ Two or more tools used together have little agreement

SAST tool Studies

1. Sandia Software Security Group, 2018
2. NIST, 2018
   - *SATE V; NIST, October 2018*
3. CAS/NSA, 2018
   - *Static Analysis Tool Study Report – Java (Phase 1), June 2018*
   - *Static Analysis Tool Study Report – C/C++ (Phase 1), June 2018*
   - *Open Source Aggregator Tool Study Report – Java, November 2018*
   - *Open Source Aggregator Tool Study Report – C/C++, November 2018*

How they work (open-source)

1. GREP
2. Lexical Analysis
3. Context Free Grammar and Parse Tree
4. Abstract Syntax Tree and Semantic Analysis
5. Control Flow Graph and Dataflow Analysis
6. Taint Propagation
7. Formal Model Checking



Metrics

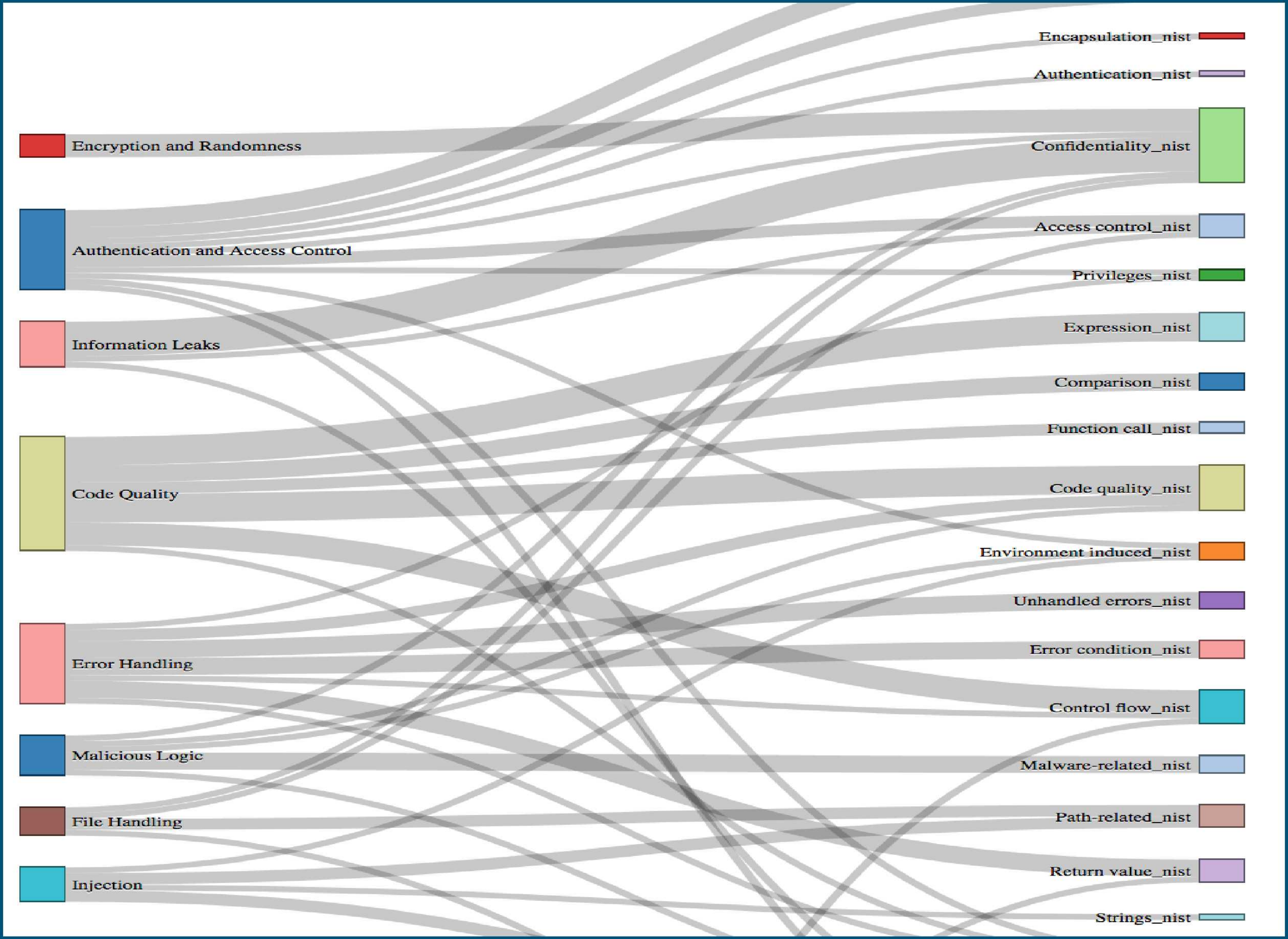
Name	Description	What's good
Coverage	Types of flaw (weakness) covered	More is better
Recall	What proportion of known weaknesses can the tool find? TP/(TP + FN)	Lower false negatives means better recall
Applicable Recall (NIST only)	What proportion of covered weaknesses can the tool find? TP/(TP + App.FN)	Accounting for weaknesses not covered improves recall
Precision	How much can I trust the tool? TP/(TP + FP)	Fewer false positives means better precision
Discrimination Rate	How smart is the tool? Compare TP on flawed cases and TN on fixed cases (#discriminations/#weaknesses)	More discriminations are better
Overlap	Which weaknesses are found by other tools?	A better overlap means more agreement among the tools

Classification

1. Low-level
   A. Common Weakness Enumeration (CWE, over 1100)
   B. Common Vulneralibity Exposure (CVE, over 16000 in 2018)
2. High-level
   A. Seven Pernicious Kingdoms (7PK + Environment)
   B. CAS ad-hoc (12 classes)
   C. OWASP top 10
   D. SANS top 25

Classification Comparison

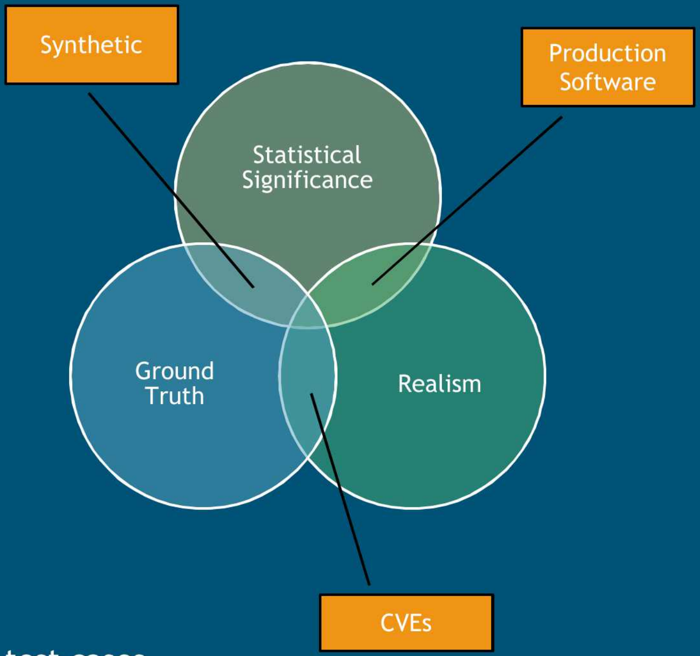
NSA



NIST

Test suites: production vs synthetic

- The SNL and NIST studies used production software[1]
  - Drawback: no ground truth
  - Advantage: realistic, statistically significent
- The SNL, NIST and CAS studies used the Juliet Suite[2] of synthetic software
  - Drawback: not realistic
  - Advantage: ground truth, statistically significant
- [Future? Bug injection
  - Realistic, ground truth AND statistically significant]



1. ~100K LOC for Java, ~500K LOC for C/C++
2. NIST, version 1.3 November 2017, 28,881 Java test cases, 64,099 C/C++ test cases

(Highly) Sampled Results (*SATE V; NIST, October 2018*)

Coverage[1] per Category for Synthetic Java.

Tool	Code Qual.	T. & S.	API	Input Val.	Error H.	Sec. Feat.	Env.	Encap.	Average
Tool L	47 %	53 %	56 %	62 %	50 %	70 %	75 %	50 %	58 %
Tool N	53 %	41 %	39 %	31 %	20 %	5 %	0 %	6 %	24 %
Tool M	41 %	53 %	33 %	4 %	50 %	0 %	0 %	6 %	23 %
Tool O	47 %	24 %	22 %	35 %	10 %	25 %	0 %	6 %	21 %
Average	47 %	43 %	38 %	33 %	33 %	25 %	19 %	17 %	

Recall[2] per Category for Synthetic Java

Tool	API	Encap.	T. & S.	Sec. Feat.	Env.	Error H.	Input Val.	Code Qual.	Average
Tool L	59 %	80 %	27 %	73 %	97 %	55 %	33 %	5 %	40 %
Tool O	26 %	35 %	18 %	25 %	0 %	4 %	17 %	2 %	15 %
Tool M	32 %	2 %	34 %	0 %	0 %	20 %	0 %	3 %	11 %
Tool N	33 %	2 %	21 %	1 %	0 %	8 %	11 %	2 %	10 %
Average	38 %	30 %	25 %	25 %	24 %	22 %	15 %	3 %	

1. Coverage = proportion of known weaknesses found
2. Recall = TP/(TP + FN)

Slide 11

CM7 Can you make these the same size? Table and fonts are different. Lower one is much easier to read.
Coram, Michael, 4/30/2019

(Highly) Sampled Results (*SATE V; NIST, October 2018*)

Overlap of multiple tools used together, finding the same known weakness

% of cases not found by any tool

Track	Participants	Number of Tools	Test Cases Found	Overlap	Proportion Found
C/C++	8	0	30 160	49 %	N/A
		1	15 663	26 %	50 %
		2	8006	13 %	26 %
		3	4279	7 %	14 %
		4	2479	4 %	8 %
		5	593	1 %	2 %
		6	191	0 %	1 %
		7	16	0 %	0 %
		8	0	0 %	0 %
Java	4	0	16 052	63 %	N/A
		1	4659	18 %	49 %
		2	2944	12 %	31 %
		3	1747	7 %	19 %
		4	75	0 %	1 %

(Highly) Sampled Results (*Open Source Aggregator Tool Study Report – Java, November 2018*)

- "The static analysis tools studied are not interchangeable. … different tools had strengths in different Weakness Classes"
- "Of the Weakness Classes studied, the tools were strongest at reporting, Pointer and Reference Handling, Initialization and Shutdown, and Buffer Handling issues. The tools were weakest at reporting Information Leaks, Authentication and Access Control, and Error Handling issues"
- "Some types of flaw were not reported by any of the tools in this study. The flaws in approximately 17% of the test cases were not reported by any of the tools studied. The Weakness Classes where test cases had the highest percentage of flaws not detected by any tool were Information Leaks (69%), Error Handling (59%), and Encryption and Randomness (45%)"

Toward Static Code Analytic Tool Orchestration

Establish Classes important to us

- Security
- Malicious code
- Synchronize of categories

Test cases building

- Create the ground truth
- Language, application

Tool Test and Categorization

- Commercial
- Open source
- Tuning
- Extension

Orchestrate tools used on

- CI/CE pipeline
- Off line analysis

SWS/Developers

- Deliver findings critical to security
- Provide actionable suggestion/solution
- Get developers on board

How to choose a SAST tool

- ❖ If a specific tool is required to obtain Authority to Operate, use that tool
  - Due to lack of overlap, using a different tool will mean adjudicating different issues at the end of the project
- ❖ If a specific tool is not required, consider open source
  - Good way to learn about SAST without substantial license costs
  - However, open source tools tend to have lower coverage and lower precision (more false positives) than commercial tools
- ❖ Customize the SAST tools to meet your project's needs
  - Start with a small set of vulnerability classes that are high risk, but with low false positives
  - Add more vulnerability classes as the team gains experience with the tools and mitigating findings
  - Consider adding custom rules to identify security issues specific to your project where applicable
- ❖ Don't rely on SAST tools alone
  - Implement the full secure software lifecycle: