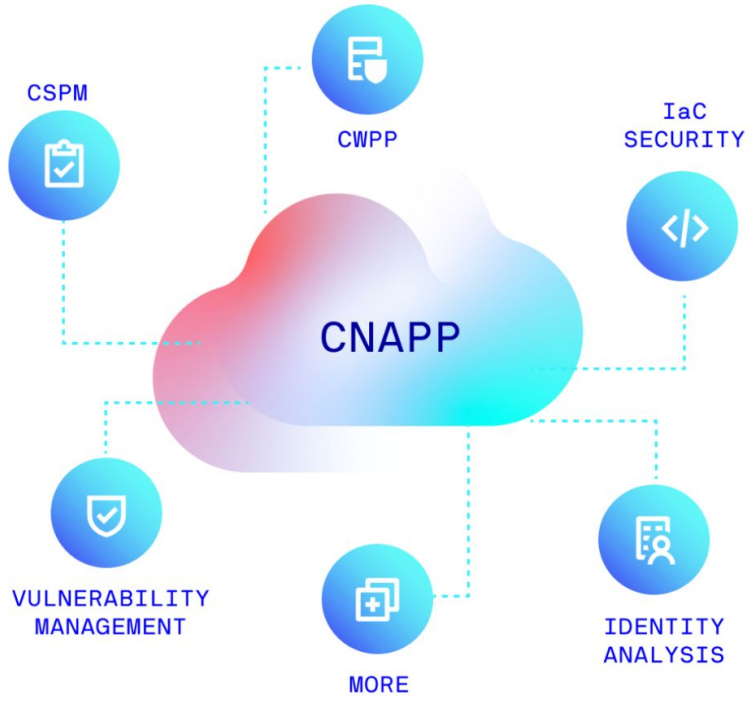
Peter,
the May and the Must,
and Lacework

Patrice Godefroid

Lacework = Cloud Security is a Data Problem



A unified cloud security platform that connects the dots for you

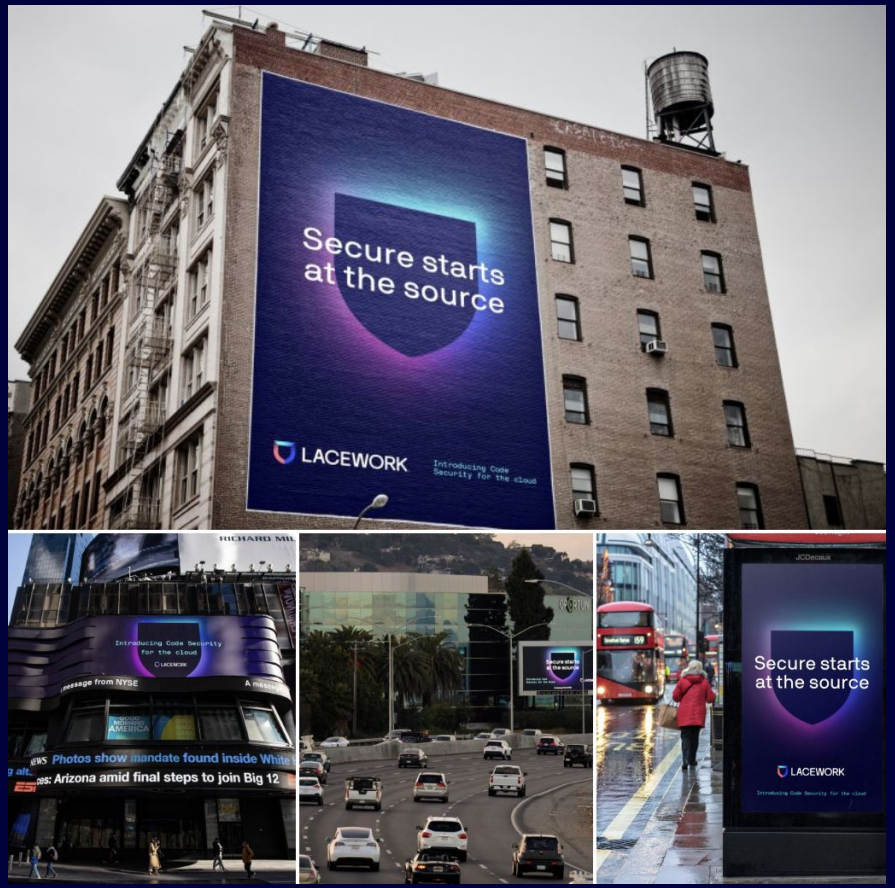
Cloud security is a data problem. Our CNAPP automatically makes sense of all your cloud data and uses your own data to better protect your entire environment — from build time through runtime.

Introducing Code Security @ Lacework [AWS re:Invent, Nov 2023]

Code Security =

- IaC (Infrastructure as Code)
- SCA (Software Composition Analysis)
- SAST (Static Analysis)
  - Quick
  - Deep

\+ Code-Aware Agents



Introducing Code-Aware Agents [RSA, April 2023]

What? Agent that detects what packages are executed (active) or not (inactive)

Why? Vulnerable packages that are inactive are **not a security risk** (if they stay inactive)

How? Monitor **all** package executions **all** the time

- Monitor file system operations
- When a package file is opened/executed, the package is declared as active
- Periodically send Active-Package data to the Lacework platform/cloud

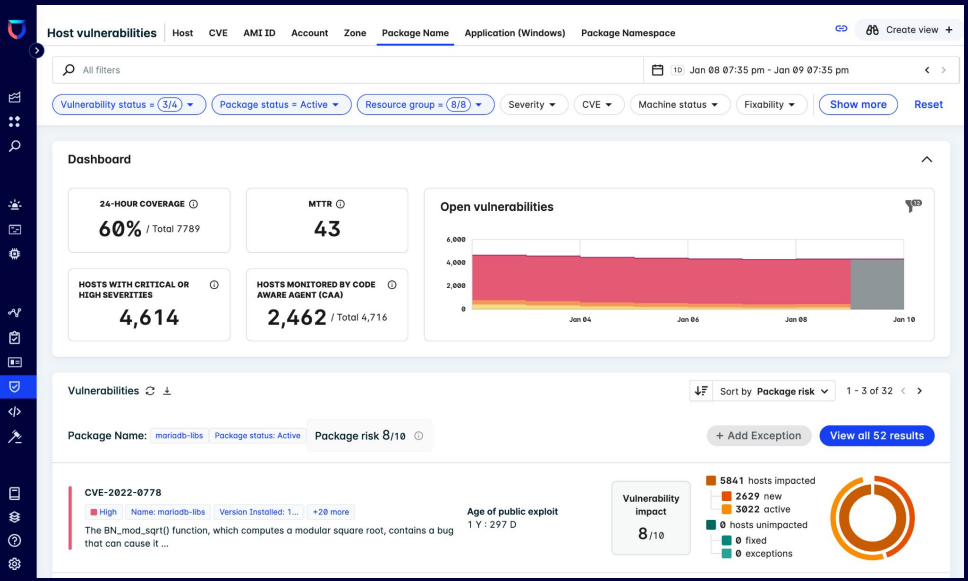
Key: low cost

- Near zero CPU+MEM agent cost
- Low-data volumes
- Reliable, safe, invisible

→ Always on, everywhere

→ No FNs and no FPs

→ PROOF of package inactivity !



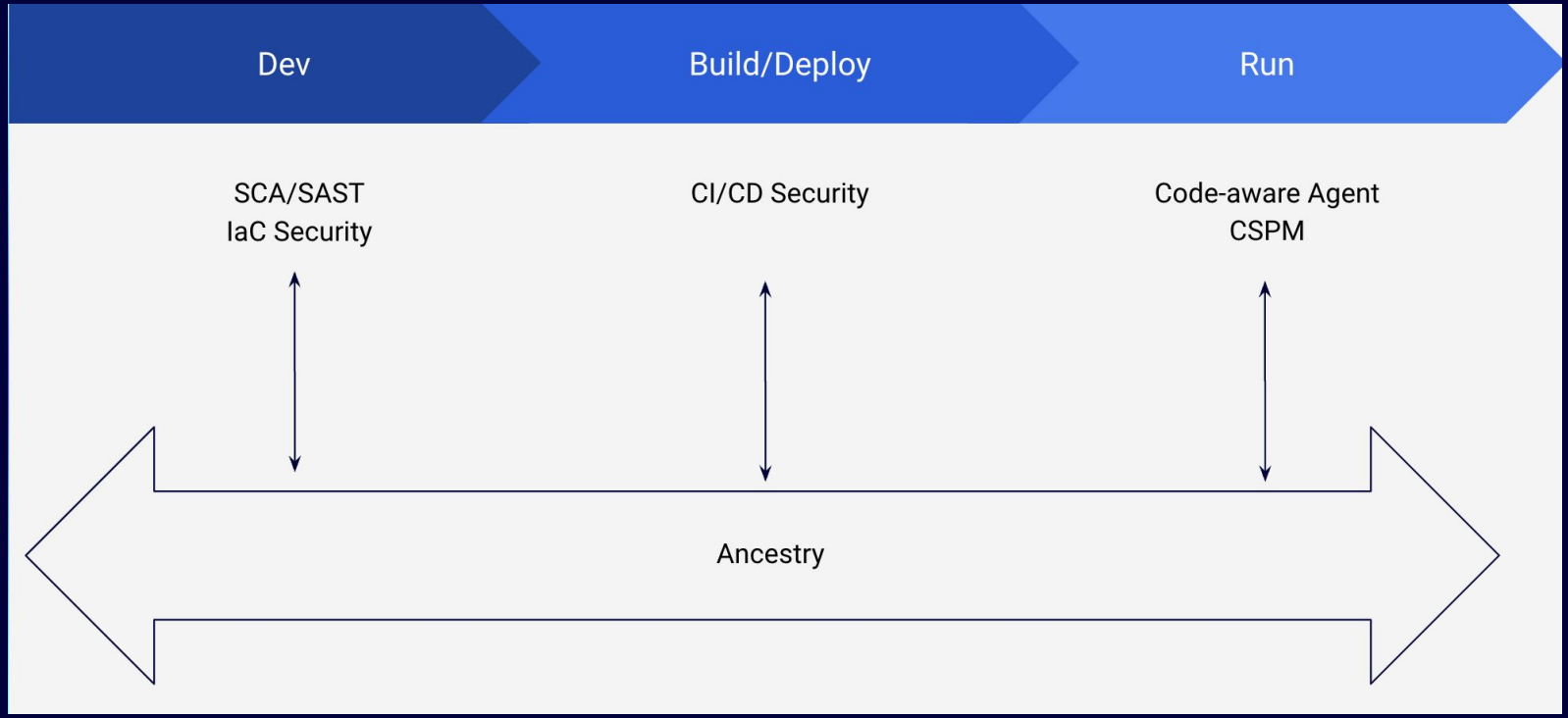
LACEWORK

Sample Results

```
Results from 2428 machines running CAA from 2023/03/20 to 2023/04/03:

Number of unique packages: 1361
Number of unique ACTIVE packages: 235
      Breakdown by severity:
            14 Critical, 33 High, 43 Medium, 8 Low, 137 Not vulnerable
Number of unique INACTIVE packages: 1126
      Breakdown by severity:
            25 Critical, 130 High, 176 Medium, 49 Low, 746 Not vulnerable

Active packages vs total packages: 235/1361 = 17%
Active vulnerable packages vs total vulnerable packages: 98/478 = 21%
Active vulnerable packages vs total packages: 98/1361 = 7%

CAA-savings = inactive vulnerable packages vs total vulnerable packages = 79%
```

Insight: Most vulnerable packages are inactive ! (from 30% to 90%, average 65%)

From Code to Cloud and Back



Claim: the Cloud provides new opportunities for Program Analysis

The May and the Must…

BUGS'2005

The Soundness of Bugs is What Matters (Position Statement)

Patrice Godefroid
Bell Laboratories, Lucent Technologies

There is one thing stronger than all the armies in the world; and that is an idea whose time has come. – Victor Hugo

In this short note[1], I argue that most program analysis and verification research seems confused about the ultimate goal of software defect detection.

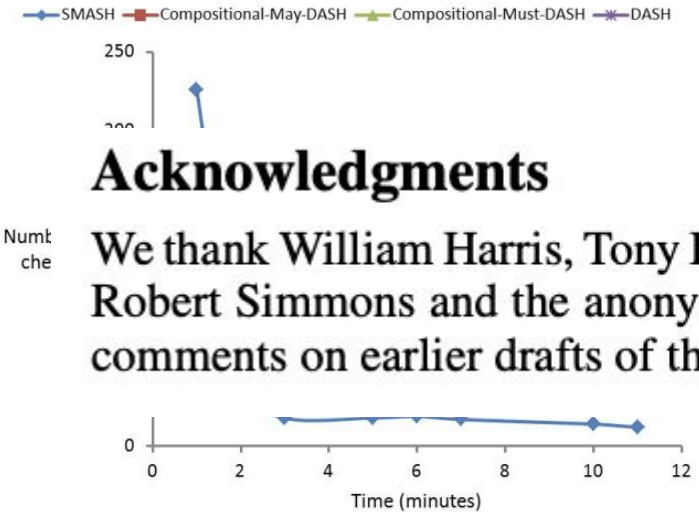
The Goal is to Find Bugs. The main practical usefulness of software defect detection is the ability to *find bugs*, not to report that "no bugs have been found". Unfortunately, the latter is sometimes confused for a *correctness proof*. In practice, there is no abstraction implies approximate reasoning. While (a) is a hard problem (i.e., often requires user assistance), (b) is simply an *engineering issue* (see below).

Alternatives. After realizing that *"The Soundness of Bugs is What Matters"*, alternatives emerge. Here are three concrete examples, drawn from my own work: **1. VeriSoft** is a software model checker for languages like C and C++ which uses a run-time scheduler for

The May and the Must…

POPL'2010

Compositional May-Must Program Analysis: Unleashing the Power of Alternation



Acknowledgments

We thank William Harris, Tony Hoare, Akash Lal, Andrew Santosa, Robert Simmons and the anonymous reviewers for their insightful comments on earlier drafts of this paper.

Program	MAY-DASH (gure 8)		COMPOSITIONAL-MUST-DASH (Figure 7 + Figure 9)			SMASH (Figure 10)			
	-outs	Time (min)	Summaries	Time-outs	Time (min)	Summaries Not-May	Summaries Must	Time-outs	Time (min)
parport		**14**	6	0	**44**	0	6	0	**14**
serial		**310**	8	7	**278**	64	7	0	**59**
serial1									**39**
sys1									**19**
sys2									**6**
flpydi									**43**
pscr2									**24**
pscr1									**26**
modem									**15**
1394vd									**26**
1394di									**115**
featur									**20**
featur		**156**	20	0	**49**	2	24	0	**14**
func_fa		**186**	21	2	**120**	5	19	0	**19**
featur		**146**	17	1	**79**	3	19	0	**8**
featur		**130**	25	3	**145**	2	25	0	**27**

Figure 11. Comparison of SMASH with COMPOSITIONAL-MAY-DASH, COMPOSITIONAL-MUST-DASH and DASH on 303 checks.

SMASH on 16 device drivers.

LACEWORK