

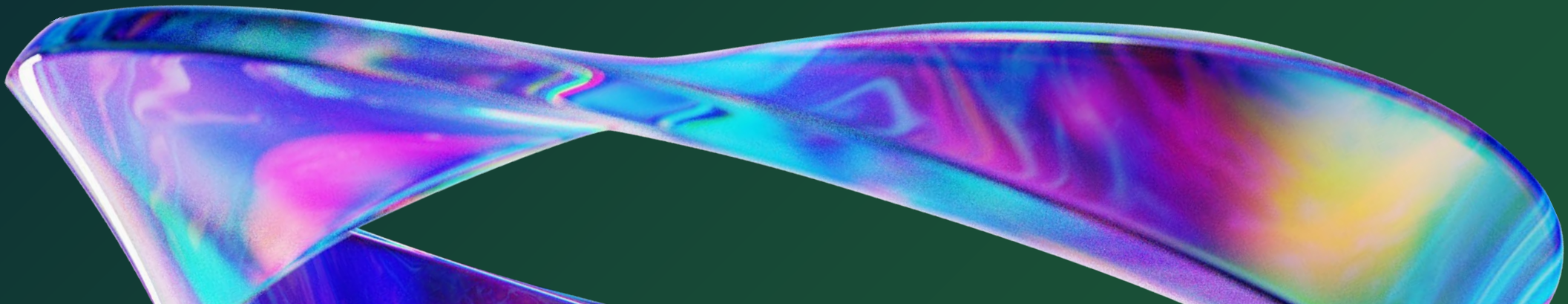
# A Developer's Guide to Making a Deal with Security

Four questions that outline how both development and security can get what they want?

Larry Maccherone

Dev[Sec]Ops Transformation, Contrast Security

[LinkedIn.com/in/LarryMaccherone](https://www.linkedin.com/in/LarryMaccherone)



**“The beatings  
will continue  
until morale  
improves”**

*Captain William Bligh  
(mutiny on the Bounty)*









node-localstorage public

DOWNLOADS **895K/MONTH**

Copyright (c) 2012, Lawrence S. Maccherone, Jr.



TRANSFORMATION.dev  
BLUEPRINT

**Larry Maccherone**  
[LinkedIn.com/in/LarryMaccherone](https://www.linkedin.com/in/LarryMaccherone)



# Developer-Centric Application Security

(aka, DevSecOps)

is

empowered engineering teams  
taking ownership of the security  
of their software

while using

Flow, Feedback, and Learning  
(aka, the 3 ways of DevOps)

to continuously

improve software value delivery

Why go DevOps?

*Speed*

*AND*

*Quality*





Question #1

Is Your Dev Team  
Ready?



Prerequisite: Start of an automated functional test suite that will grow until you trust it to prevent an “unworthy” artifact from getting to the next higher-level branch

- A single E2E test gets you as much as 30% test coverage and that’s all you need for this prerequisite
- Only use solitary unit testing for logic and libraries that are easily isolated w/ little to no mocking. Even Martin Fowler now advocates for “sociable unit tests”
- This functional test suite must be completed prior to the pull-request merge decision

# Why?

Why are functional tests important for security?

1. Because you need it for **Software Composition Analysis (SCA)**. You can't be sure that upgrading to the latest version will not break your app without testing it.
2. Because it maximizes the benefits of using **IAST tools** rather than slower and less accurate SAST tools.

Quality reasons should be enough to motivate test writing but with these security reasons, such work is now twice as valuable



# HOW?

How do you get your development teams to write tests?

1. Team working agreement document (aka definition of done, Kanban entrance criteria). Connect with me on LinkedIn for the document I start all teams with.
2. Devs write happy path test(s), at least, when adding functionality pre-pull-request even if QA writes more later
3. Of course, install a check to block the merge if a test fails, but also block if coverage ever goes down
4. Offer a pairing service to get a hello-world test written and the merge blockers installed



Prerequisites:

- Ephemeral build and test infrastructure
- Parallelized test suites so they finish in time to inform a pull/merge request merge decision (typically 5 minutes)

# The pull request is the ideal place to surface feedback



Developers are proud of the code we wrote this morning



We want nothing more than our teammates to say how wonderful it is by merging it into the next higher-level branch that afternoon



We will jump through hoops to make sure it looks as wonderful as it can, including taking a quick lesson on the found vuln



Days later, we've already moved on to a new heart throb... the code we wrote that morning





Question #2

What does a good  
developer-centric approach  
look like?





Why not just use...

NIST, SANS, OWASP, and/or PCI,  
etc.?

- Don't all say the same thing
- Target security specialists in both terminology and approach
- Fail to specify adaptations needed for a dev-first approach
- Too many practices = overwhelming
- Not prioritized
- Fail to consider your unique context

So instead...

Lesson:  
Borrowing practices from  
industry standards without  
transformation and prioritization  
is a recipe for frustration and  
false starts

# Host a workshop to create a prioritized set of practices for your context



Bring security and the most respected development leaders into a room for a half-day workshop



Collaboratively with post-it notes on the walls (or the electronic equivalent) agree on a prioritized set of practices



Identify the work necessary to provide “the easy button” so dev teams can self-service adopt each practice



Coach dev teams with the practices from this list where you have “the easy button”



Learn from coaching sessions and dev team adoption difficulties to improve the list and each individual practice



# Example list of weighted practices

don't simply adopt this one

## A. Non-security engineering practices (DevOps)

1. (5%) Working agreements document (Kanban entrance criteria, definition of done, etc.)
2. (15%) Ephemeral build and test infrastructure [prerequisite for B.1-B.4 practices below]
3. (5%) Minimal e2e testing coverage >25% [prerequisite for IAST]
4. (3%) Robust e2e testing coverage >80%

## B. Security practices done by the development team

1. (12%) Critical clean for the 3<sup>rd</sup> party code you import (aka SCA, SBoM)
2. (12%) Critical clean for for the 1<sup>st</sup> party code you write (SAST or IAST)
3. (7%) High clean 3<sup>rd</sup> party
4. (7%) High clean 1<sup>st</sup> party
5. (10%) Secrets management – No secrets in source code repositories
6. (3%) Incremental/Agile Threat modeling using OWASP Cornucopia
7. (10%) Runtime application self-protection (RASP)

## C. Security practices done with the help of resources external to the team

1. (5%) Threat modeling done with a security architecture expert on cadence determined by risk
2. (10%) Pen testing on cadence determined by risk

## Critical elements

1. Starts with non-security practices. Emphasizes engineering excellence & makes security practices efficient and effective
2. Prioritization and Gamification
3. No “points” for scanning. Only for getting and staying “clean”
4. Shallow onramp. Ex: severity, 1<sup>st</sup> vs 3<sup>rd</sup> party, but there are alternative ways to slice it



Question #3

What is the criteria for a  
good dev-centric App/API  
security tool?





# The need for speed

## Criteria #1: Speed

- In order for the results to come back in time to inform a pull request merge decision, it must finish fast. **5-minutes** is a good target but up to 15 minutes may be acceptable
- Most SAST tools take much longer than that
- SAST tools with an incremental mode or a demand-based algorithm can be faster
- **IAST tools** are as fast as your test runs (plus no more than 15%) so are a perfect fit for parallelized test suites



## Criteria #2: Accuracy

- Every false positive wastes your time. 30% to 80% of untuned SAST tool findings are false positives
- Every false positive erodes your trust in a tool
- False negatives leave you with unknown risk
- The highest scoring SAST tool, Contrast Scan, only scores in the high 50's on OWASP's accuracy benchmark
- Contrast Assess, an IAST tool, scores 100 on that same test
- You can never auto-gate unless you have high accuracy



# Favor IAST

There are no parallelized SAST tools on the market and for any app that's big enough to call for parallelized test suites, full SAST scans take too long to run pre-pull-request merge decision.

However, IAST tools are up to 36x faster and automatically parallelized when you split up your test suite.



Question #4

How does security “coach”  
adoption of these tools  
and practices?





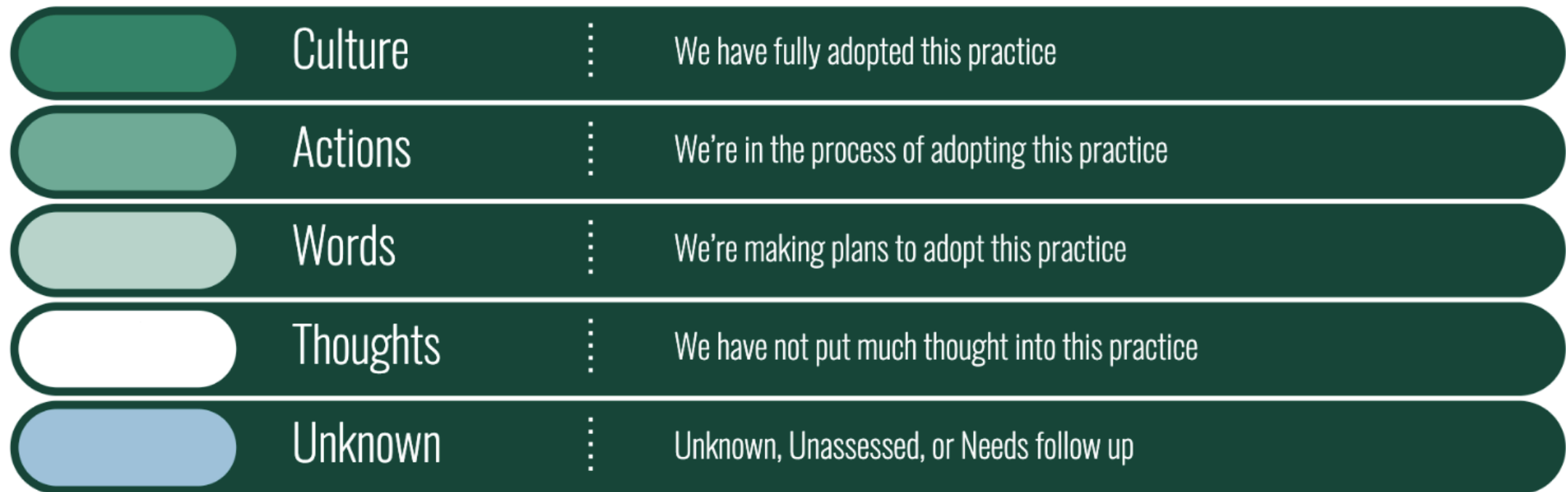
Lesson:  
Coach is the first new key role

Lesson:  
Coach gradual improvement and  
get to “culture” on a few  
practices out of a weighted list  
before expanding

# Practice adoption states

## Thoughts → Words → Action → Culture

### (ThWAC)



# Gradual practice maturity growth via coaching



A coaching workshop for each two-pizza development team



Partial gap analysis starting with most valuable practices first



Shift into planning mode as soon as the gap analysis identifies 1-3 practices the team wants to get to “culture” in the next 90 days



Coach for 90 days



Re-sync workshop starting with the next round of most valuable practices



# Transformation Blueprint Visualizations

TRANSFORMATION.dev  
BLUEPRINT

Culture

We have fully adopted this practice

Actions

We're in the process of adopting this practice

Words

We're making plans to adopt this practice

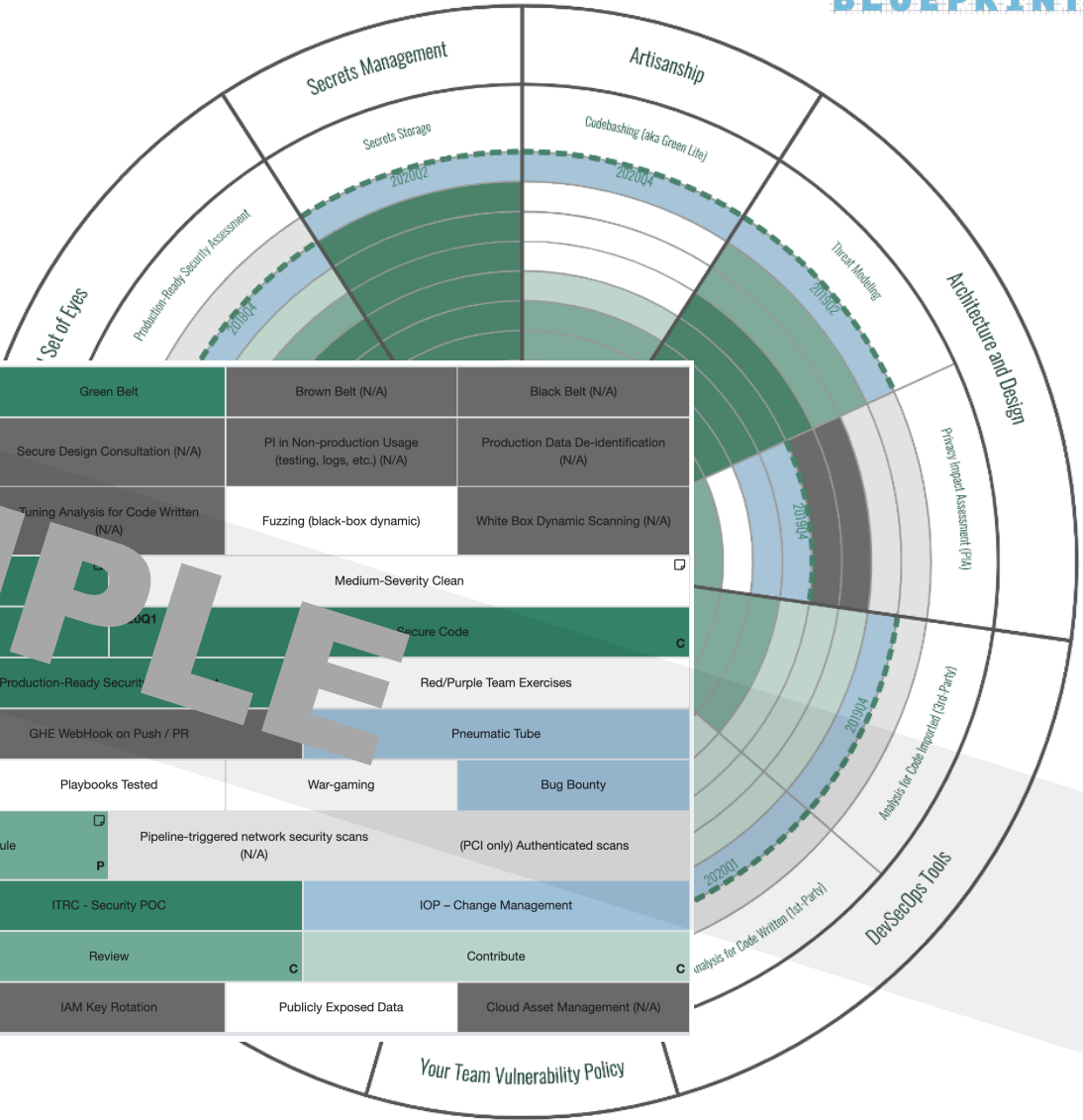
Thoughts

Unknown

Deferrals

Trade-offs

Artisanhip	Low Belt	2020Q4	Codebashing (aka Green Life)	Green Belt	Brown Belt (N/A)	Black Belt (N/A)
Architecture and Design	2019Q2	2019Q4	Security Impact Assessment (PIA)	Secure Design Consultation (N/A)	PI in Non-production Usage (testing, logs, etc.) (N/A)	Production Data De-identification (N/A)
DevSecOps Tools	2020Q1	2020Q2	Analysis for Code Written (1st Party)	Tuning Analysis for Code Written (N/A)	Fuzzing (black-box dynamic)	White Box Dynamic Scanning (N/A)
Your Team Vulnerability Policy	2020Q2	2020Q4	High-Security	Medium-Security	Medium-Security Clean	
Vulnerability Working Agreements	2020Q4	2020Q1	Team-external/Comcast-internal Vulnerabilities	Secure Code		
Second Set of Eyes	2020Q4	2018Q4	Security Peer Review	Production-Ready Security	Red/Purple Team Exercises	
Secrets Management	2020Q2		Secrets Storage	GHE WebHook on Push / PR	Pneumatic Tube	
Incident / Public Vulnerability Response Capability	Awareness	Playbook	Playbooks Tested	War-gaming	Bug Bounty	
Network-originated Scans	Scan findings resolved	Patching Schedule	Pipeline-triggered network security scans (N/A)	(PCI only) Authenticated scans		
Asset Management	ITRC - Asset Relationships	ITRC - Security POC	IOP - Change Management			
Reference Components and Designs	Use	Review	Contribute			
Cloud-Specific Practices	Cloud-Specific Design Patterns	IAM Key Termination	IAM Key Rotation	Publicly Exposed Data	Cloud Asset Management (N/A)	





# Coaching Individual Teams Scales

Each workshop is 1.25 hours on average. If a coach does 10 a week, that's only 12.5 hours leaving the rest for ad-hoc coaching. There are 13 weeks each quarter but allowing for PTO, etc. let's say 10.

$10 \times 10 = 100$  teams per coach

Lesson:  
Not only does coaching individual teams scale, it's a 4x cost savings for 6x improvement in cyber risk reduction, so 24x more value

# TRANSFORMATION.dev BLUEPRINT

Open sourced Dev[Sec]Ops practice blueprints and coaching tool

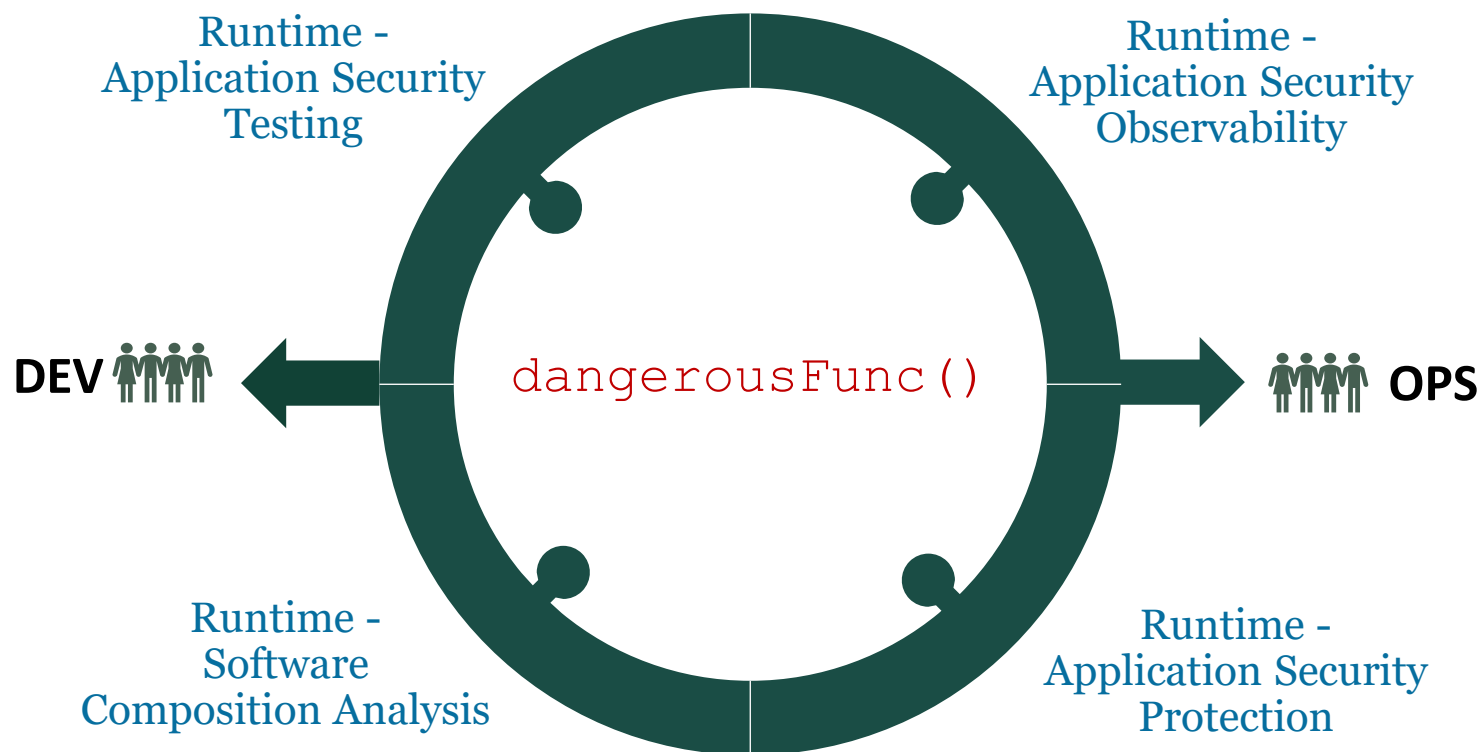
<https://transformation.dev>

# The deal!



- **Development** gets:
  - Delightful tools; and
  - A prioritized list of easy-button practices tailored to your way of working
- **Security** gets:
  - A commitment from each development team to adopt one to three of those practices, in roughly the priority order, over each successive 90-day period until the key 8-12 become "culture"

# Simplifying by securing from within with **Runtime Security**



**Do this one thing and get all this capability:**

## **R-AST (IAST)**

Find vulnerabilities in the 1<sup>st</sup> party code you write

Replaces SAST and DAST

## **R-SCA**

Find truly exploitable vulnerabilities in the 3<sup>rd</sup> party code you import

Replaces SCA/SBOM/OSS

## **R-ASO**

Reveal the security blueprint of your distributed systems

Complements Observability

## **R-ADR (RASP/ADR)**

Detect attacks and prevent exploit attempts

Strengthens your WAF



# What's next?

- Questions?
- Connect: <https://LinkedIn.com/in/LarryMaccherone>
  - Ask questions
  - Get a copy of slides
  - Learn more about security tools that appeal to developers
  - Schedule a Dev[Sec]Ops Transformation workshop at your organization

