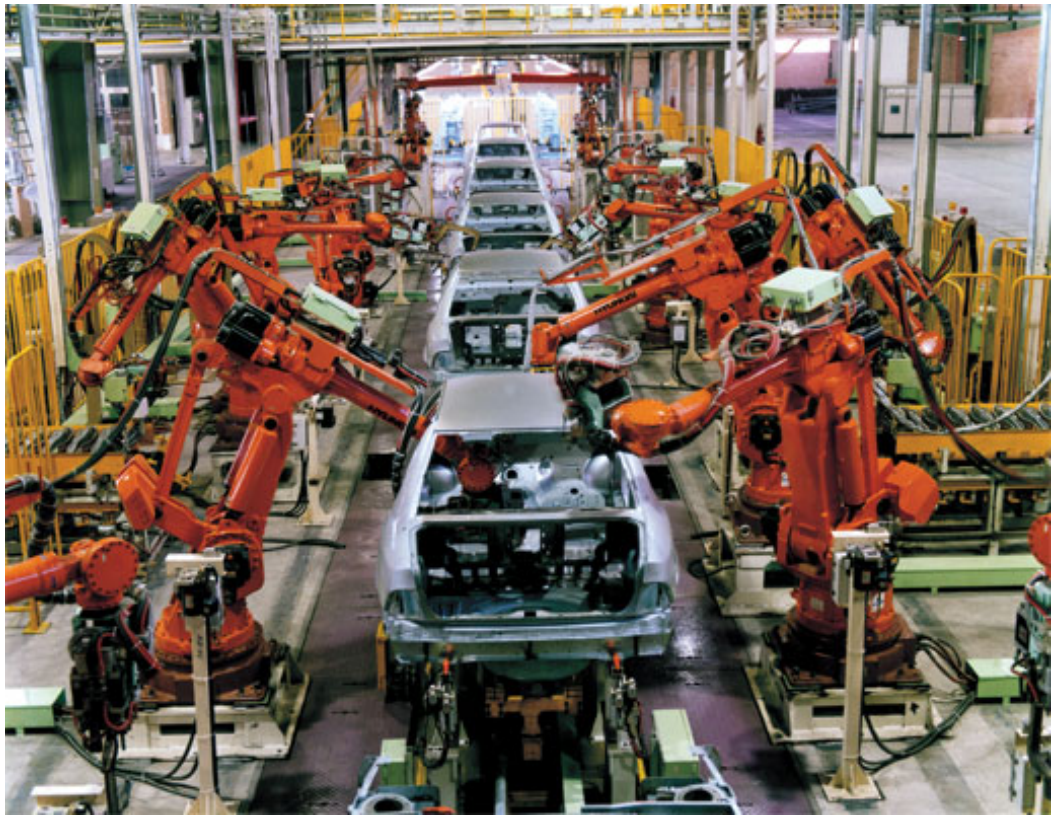
The Hidden Risk of OSS

The Dawn of Software Assembly



The Language of Security is Risk

What is Risk

+

=

"…WE OWE A DUTY OF REASONABLE CARE TO OUR NEIGHBOR"

Lord Atkin: Donoghue v. Stevenson (1932)

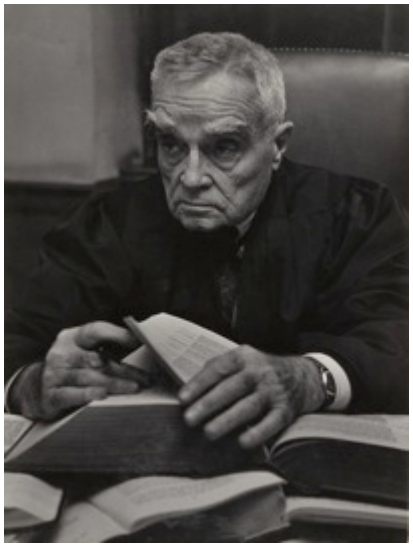
"…a manufacturer of products, which he sells in such a form as to show that he intends them to reach the ultimate consumer in the form in which they left him with no reasonable possibility of intermediate examination, and with knowledge that the absence of reasonable care in the preparation or putting up of products will result in an injury to the consumer's life or property, owes a duty to the consumer to take that reasonable care."

"It (Buick) was not at liberty to put the finished product on the market without subjecting the component parts to ordinary and simple tests....The obligation to inspect must vary with the nature of the thing to be inspected. The more probable the danger, the greater the need of caution."

MacPherson v. Buick Motor Company, 217 N.Y. 382, 111 N.E. 1050 (1916)

Justice Benjamin N. Cardozo

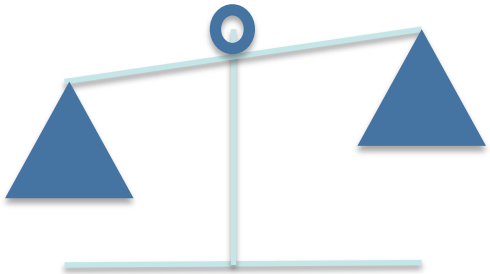
"...IF THE PROBABILITY BE CALLED P; THE INJURY, L; AND THE BURDEN, B; LIABILITY DEPENDS UPON WHETHER B IS LESS THAN L MULTIPLIED BY P: I.E., WHETHER B < PL".



United States v. Carroll Towing Co.
159 F.2d 169 (2d. Cir. 1947)

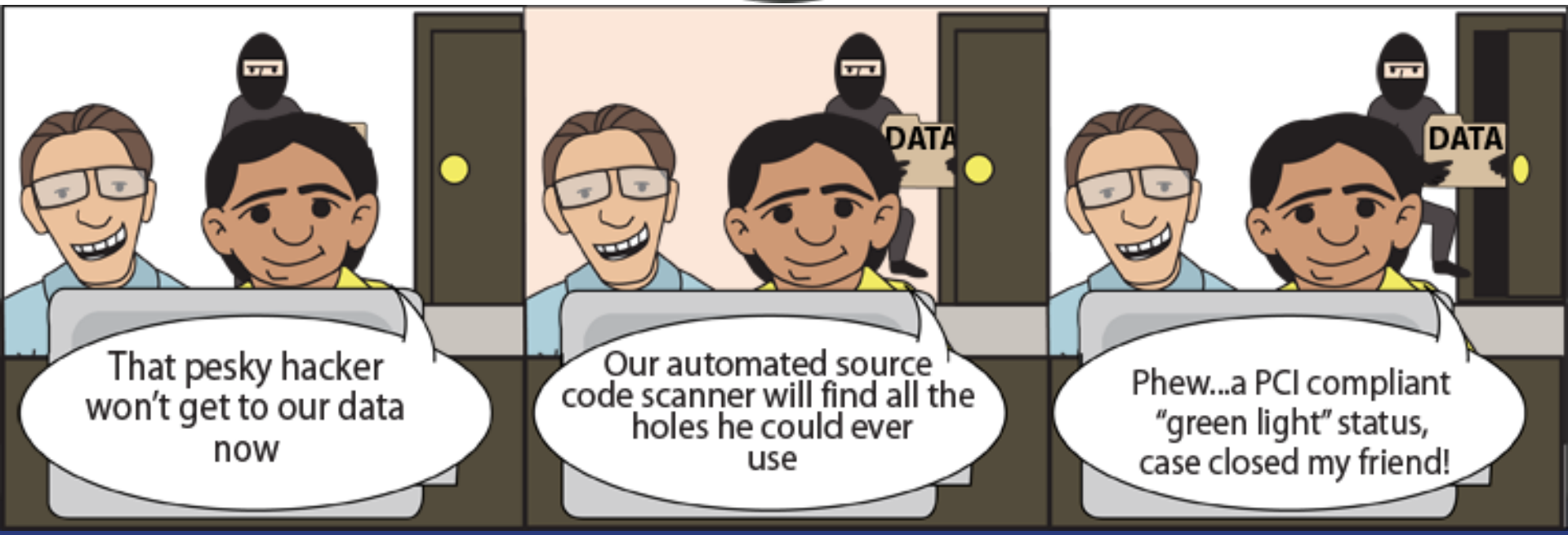
Translation: If the Cost of Protecting Against Harm is less than the Cost of the Damage Multiplied by the Likelihood of the Damage, then there is **negligence.**

Risk = probability x impact



Security concerns are across the Enterprise

Development	Operations	Security
Features	Performance	Security
Usability	Reliability/Scalability	Compliance
Performance	Compliance	Everything Else
Reliability/Scalability	Security	
Maintainability	Maintainability	
Security	Features/Usability	
Compliance		


That pesky hacker won't get to our data now
DATA
Our automated source code scanner will find all the holes he could ever use
DATA
Phew...a PCI compliant "green light" status, case closed my friend!

Prevention	Detection	Monitoring
Firewall	IDS	SIEM
Encryption	SAST	DAM
IPS	DAST	RAST
WebApp Firewall (WAF)		

Evolution of Spend

Figure 1. Magic Quadrant for Dynamic Application Security Testing

Source: Gartner (December 2011)

DAST is a very mature market, but isfocused primarily late in the development cycle and not integrated into development.

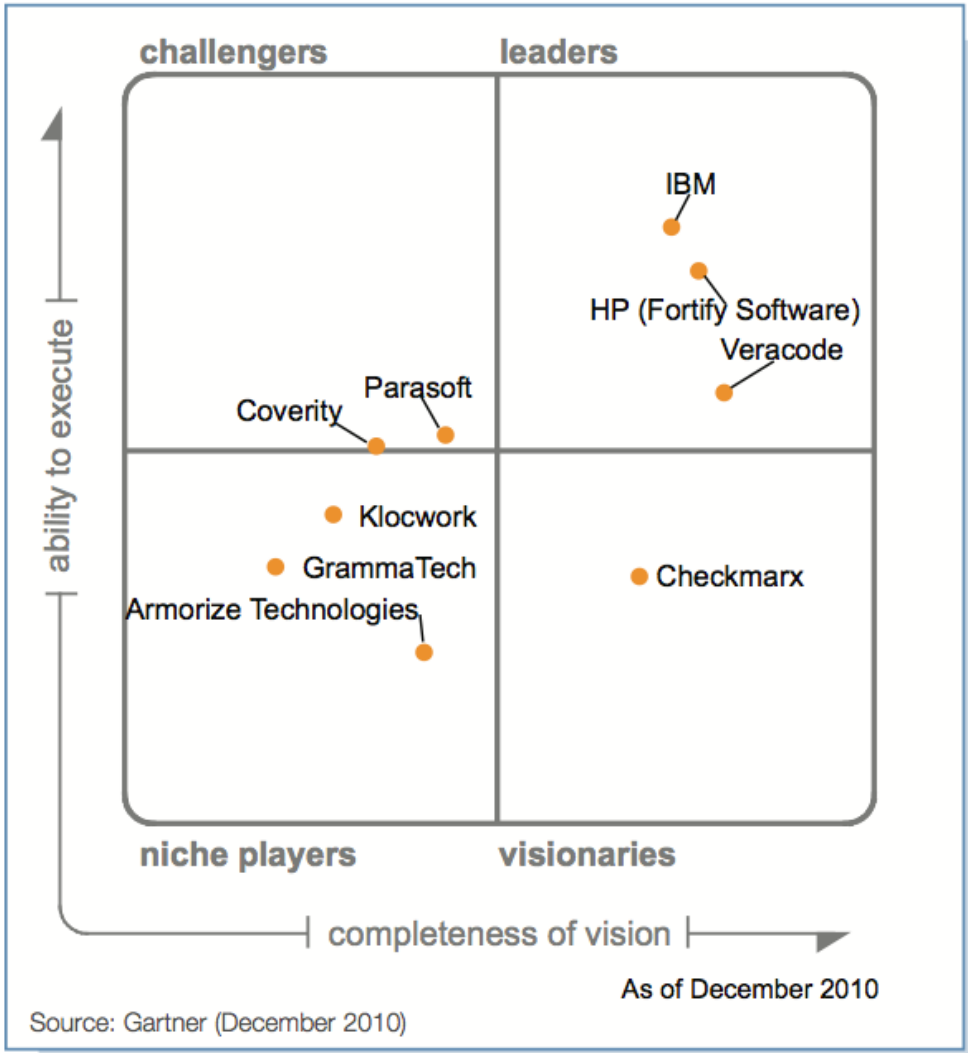
Pros

- Finds exploitable issues
- Mostly language agnostic
- Finds some infrastructure issues

Cons

- Often requires complex configuration
- Accuracy drops for non-reflected issues
- Used late in SDLC

Figure 1. Magic Quadrant for Static Application Security Testing



Source: Gartner (December 2010)

SAST is a mature market, but is under represented outside of financial, health/insurance and retail markets.

Pros

- Can be leveraged early in the development lifecycle
- Can find issues not found using any DAST

Cons

- False Positives
- Requires security training to use effectively.
- Scanning varies from hours to days for large applications.

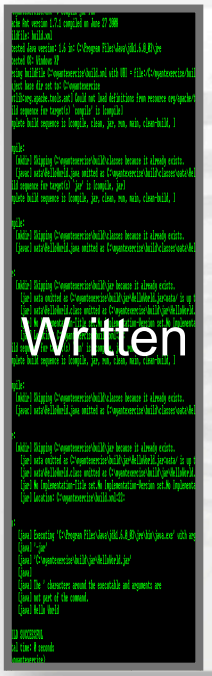
Over the past decade there are have been two predominant security technologies focused at application security.

- DAST – Dynamic Application Security Testing (Blackbox)
- SAST – Static Application Security Testing (Whitebox)

Over the last couple of years a third as emerged but has not gained significant adoption

- RAST – Runtime Application Security Testing (Glassbox)

A Sea Change in Application Development



Written

Assembled **90%**

Source: 2012 / 2013 Sonatype analysis of more than 1,000 enterprise applications

The Ice-Caps are Melting

Presentation Layer

Business Logic

Component Layer

Proprietary Components

3rd Party Components
(Libraries, Frameworks, Utilities & Other Components)

Database

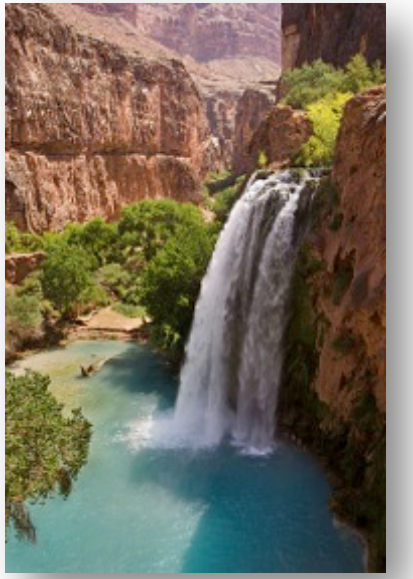
Operating System

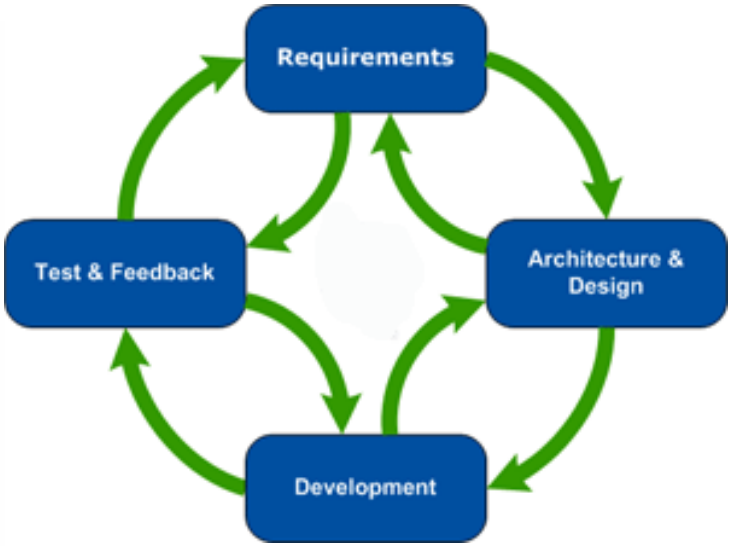
Firmware

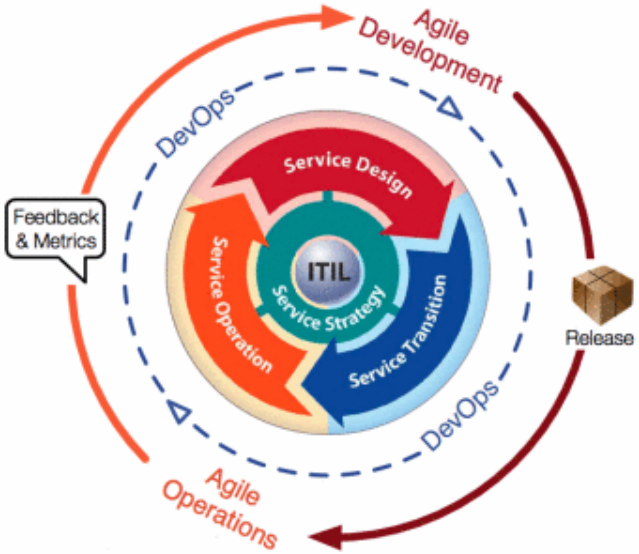
Network

Development Must Keep Up with Pace Of Innovation

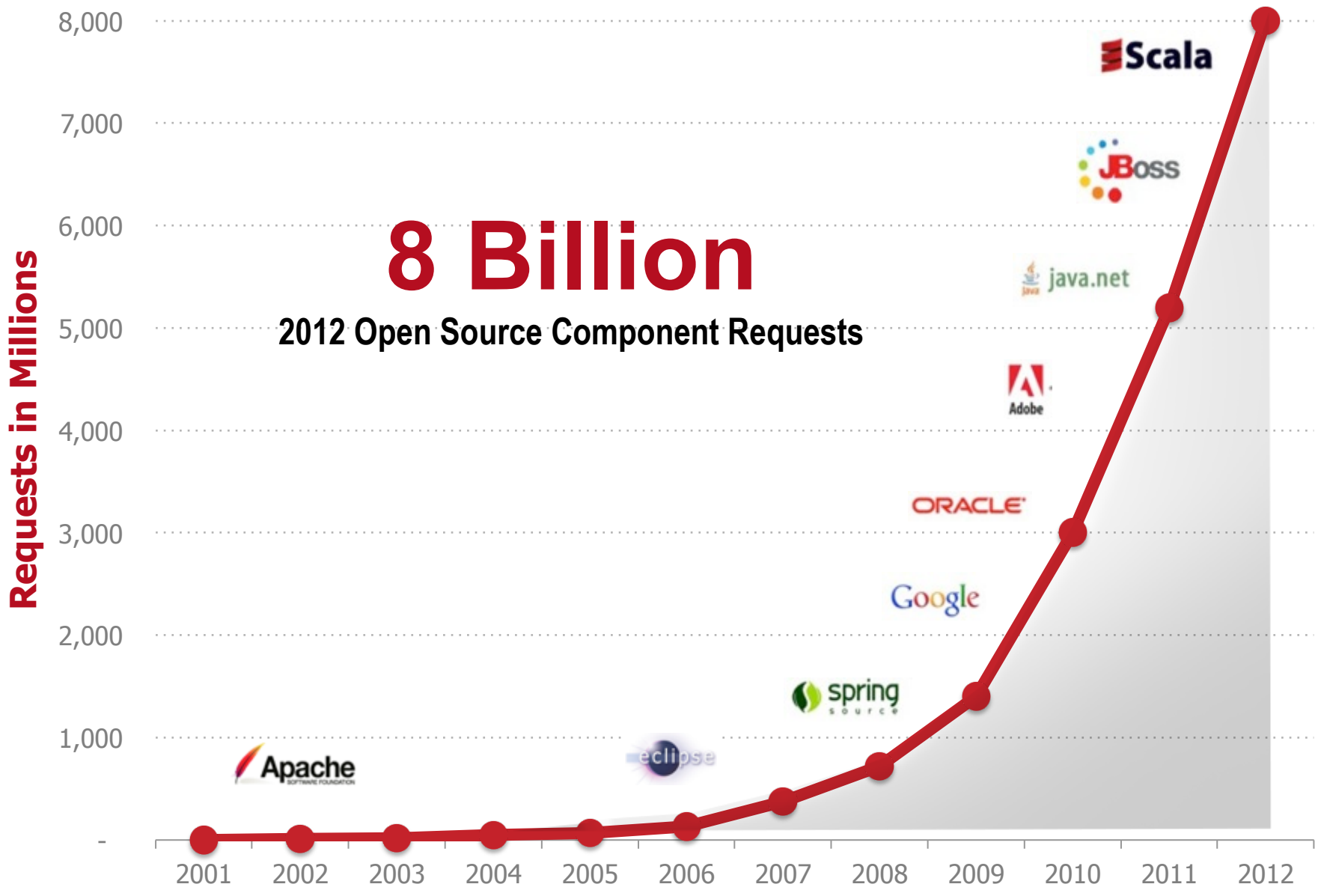
Development must change



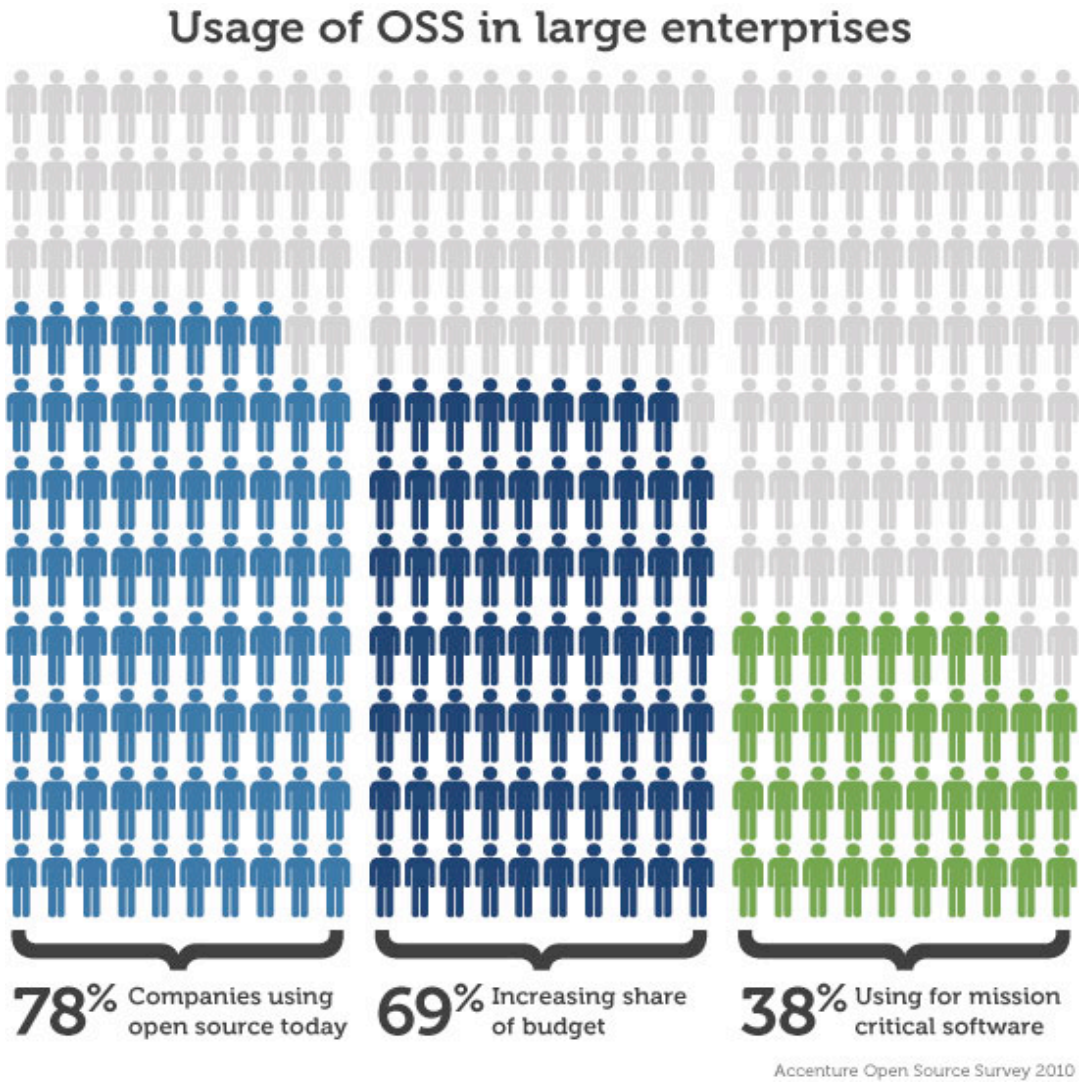




Component Usage Has Exploded

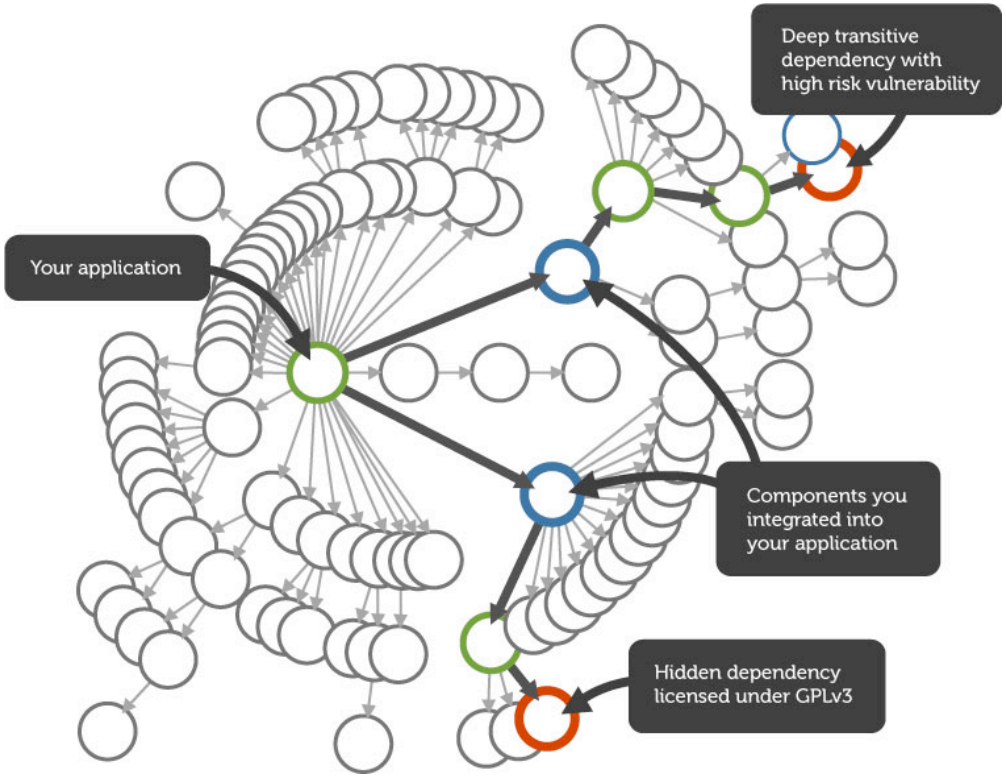


“But we don’t use Open Source”



It’s no longer a question of whether you use OSS, it’s how many components are being used & where

No “Throat to Choke”



- Discovering a security issue is half the battle
- Transitive and hidden dependencies make it extremely difficult to assign responsibility to propagate fixes throughout the component chain

A Highly Complex Ecosystem

Complexity

One component may rely on 00s of others

Diversity

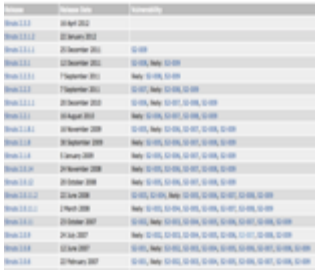
40,000 Projects
200MM Classes
400K Components

Volume

Typical Enterprise Consumes 000s of Components Monthly

Change

Typical Component is Updated 4X per Year



No Visibility

No visibility to what components are used, where they are used and where there is risk

No Control

No way to govern/enforce component usage. Policies are not integrated with development .

No Fix

No efficient way to fix existing flaws.

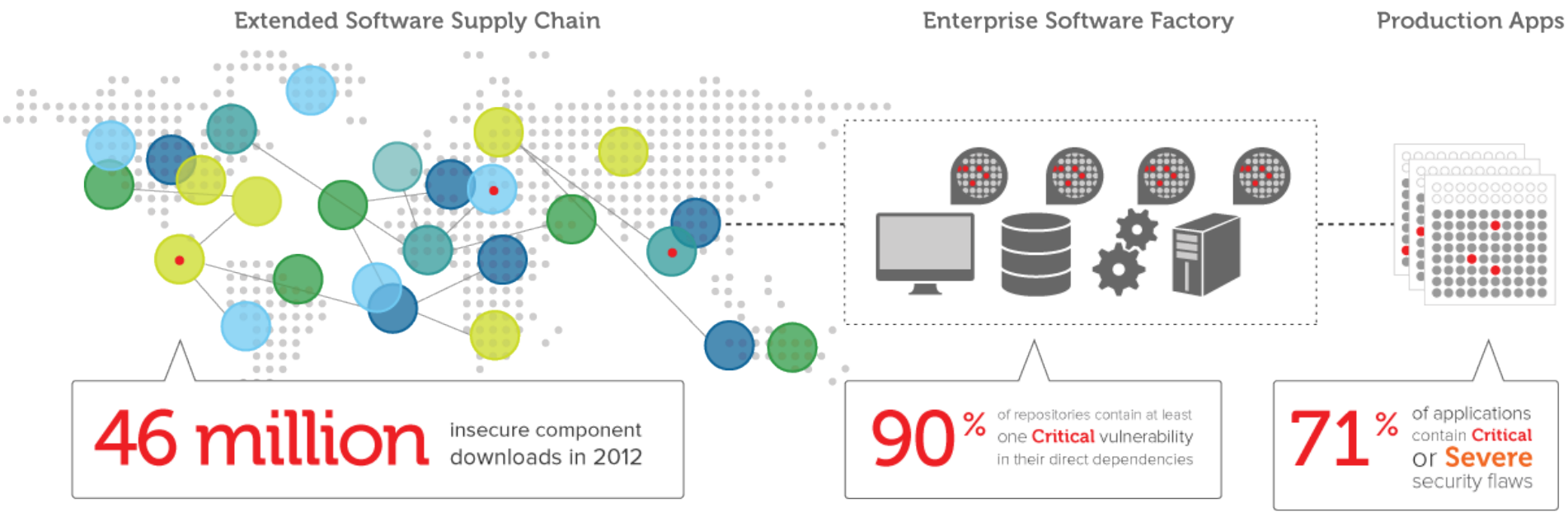
A Massive Supply Chain Problem

46 Million Insecure downloads in 2012

18 Thousand organizations downloaded Struts framework with "severe" security flaw

4 Thousand organizations downloaded Struts 1.x with known security flaws

The Practical Reality



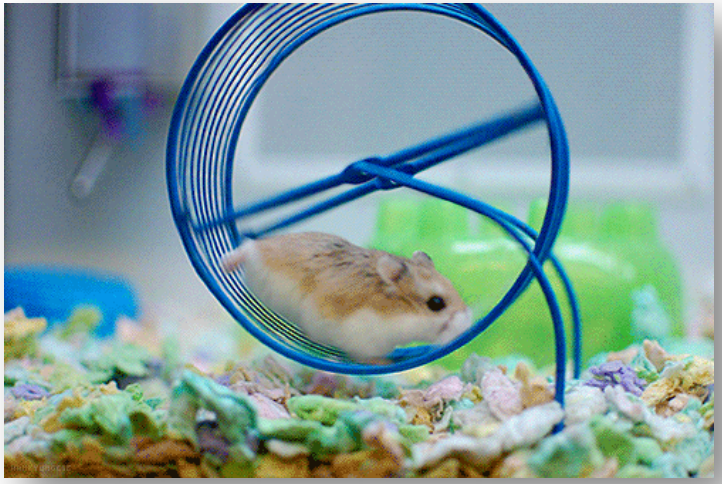
Success Requires Discipline

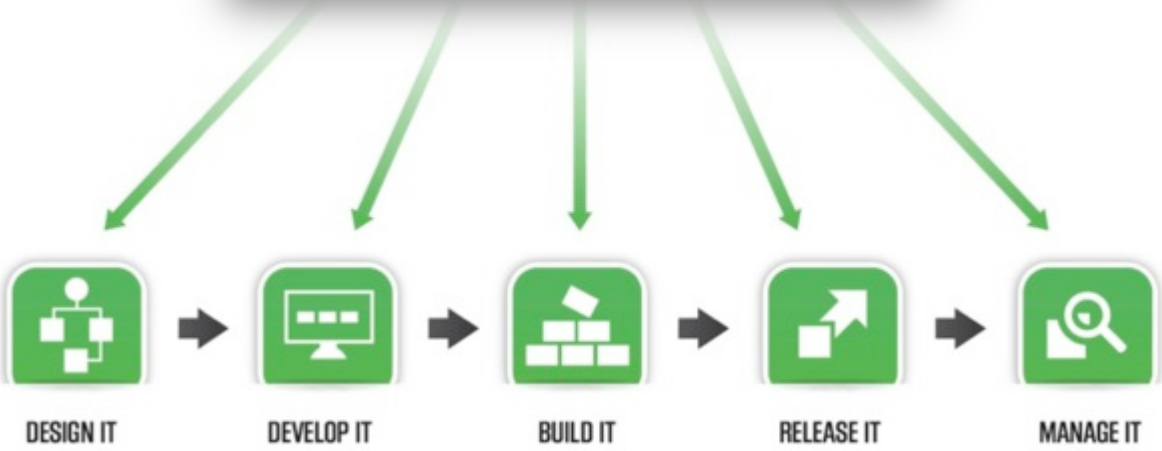
The Problem is Not Problem Discovery

- When our software development ecosystem looks like this it is easy to find problems
- The real challenge is to develop at scale and deliver continuous value continuously when everything else is a mess

Go Fast, Be Secure

“Haven’t I heard this story before?”





Component Lifecycle Management

1

Secure Consumption
with the use of certified components & integrity checking throughout the lifecycle

2

Govern Development
to ensure policy compliance without disrupting developer productivity

3

Profile Exposures
to proactively identify and prioritize action

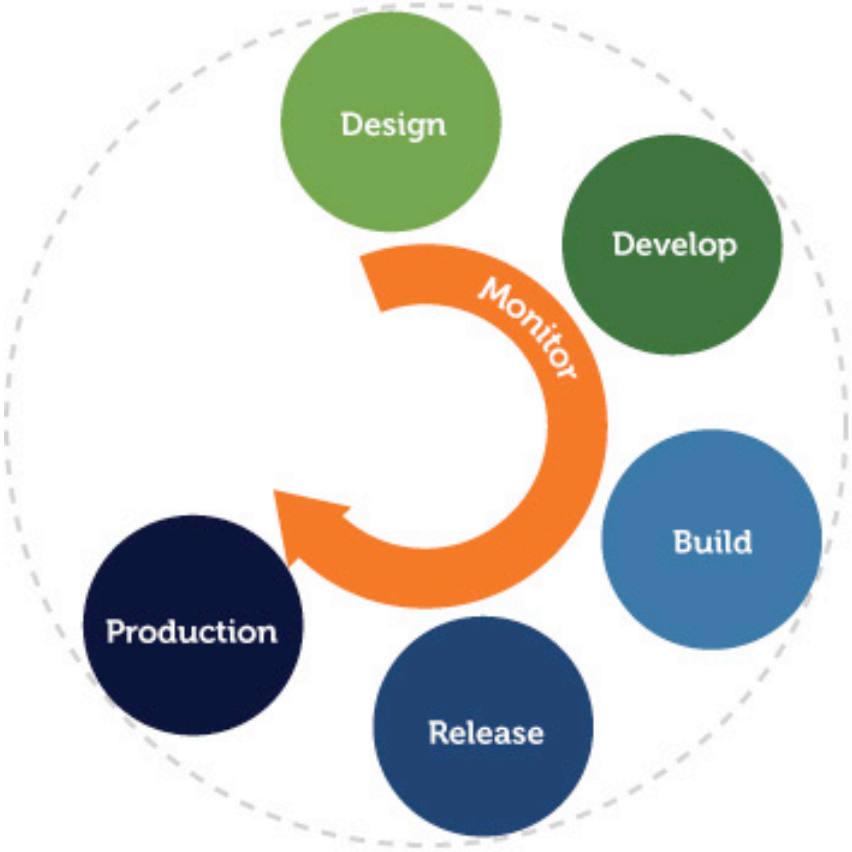
4

Remediate Risk
by preventing & quickly fixing security & IP vulnerabilities

5

Monitor Threats
in production applications to ensure continuous trust in critical operations

You Have to Ask the Right Questions



Q How do you choose components to include in your application?

A Thoughtfully select and identify components using quality, security, and licensing information.

Q How do your developers know what components to use, and when they should upgrade?

A Provide your team with real-time information and updates directly within the tools they use every day.

Q Do you monitor and control what makes it into a build?

A Enforce policy through your build and continuous integration infrastructure.

Q Do you know your full bill of materials?

A Develop and maintain component inventory for every application.

Q Do you know when vulnerabilities are found in deployed components?

A Monitor component bill of materials for new security flaws and identify applications for critical updates.

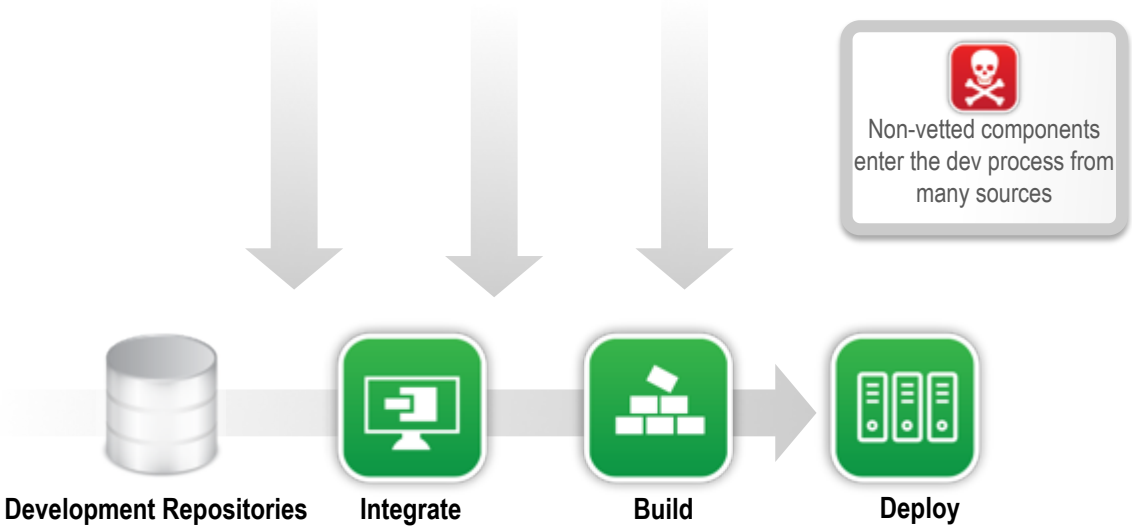
Q Do you have global visibility into open source usage?

A Know how, when, and where components are consumed organization-wide to identify risks before they become a problem.

Components Can be Compromised

Component Repositories

Google



Automated Policy Management Throughout the Lifecycle

Centralized policy administration simplifies enterprise management

Lifecycle appropriate actions enforce policy automatically

- Need to recognize that the priorities are different
- Tooling needs to adopt the practice of the practitioner not the other way around
- A Tool is not a process and a process is not a tool learn to leverage both.

Go Fast.
Be Secure.

Build security in from the start

Enforce policy in the tools you already use

Reduce risk by automating governance throughout the lifecycle

Reduce cost by fixing early in the process

React to new threats by knowing what they are and where to fix them

Go fast by using tools your developers already know

Thank You!