

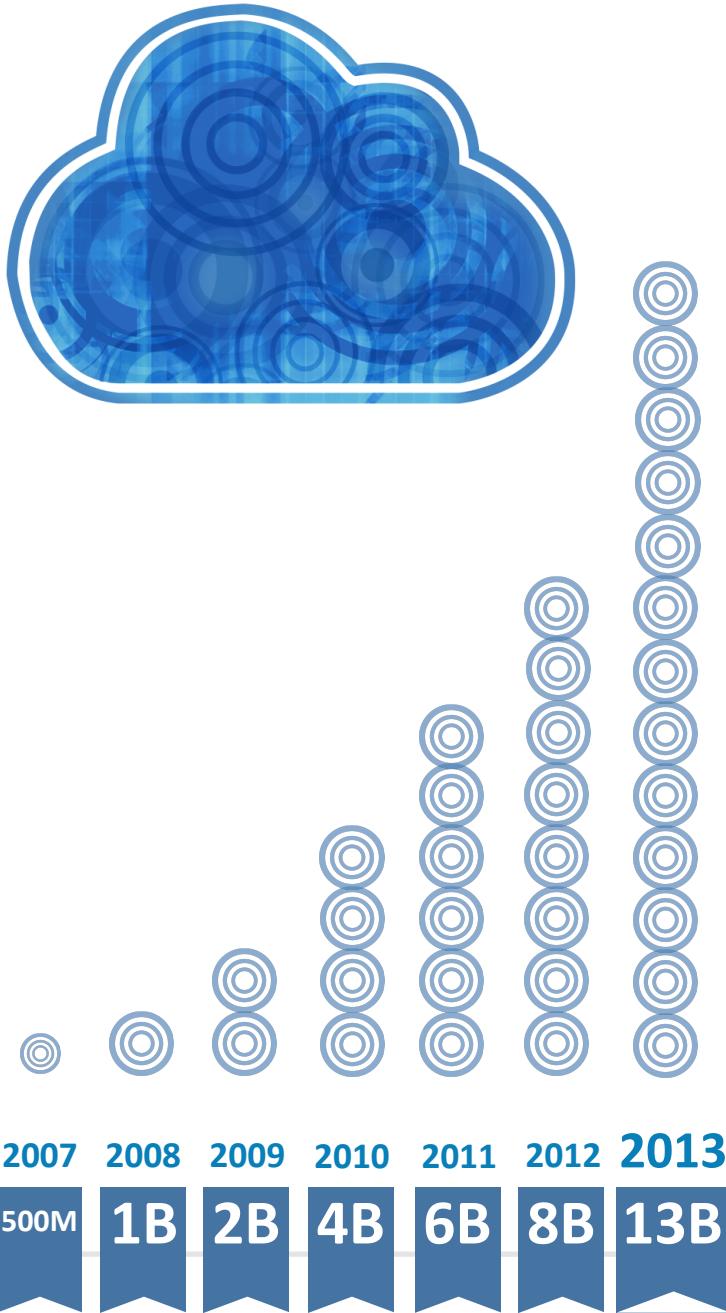


Open Source components fuel agile  
development

**Supply chain solutions for modern development**  
Brian Fox @brian\_fox

# INDUSTRIAL EVOLUTION



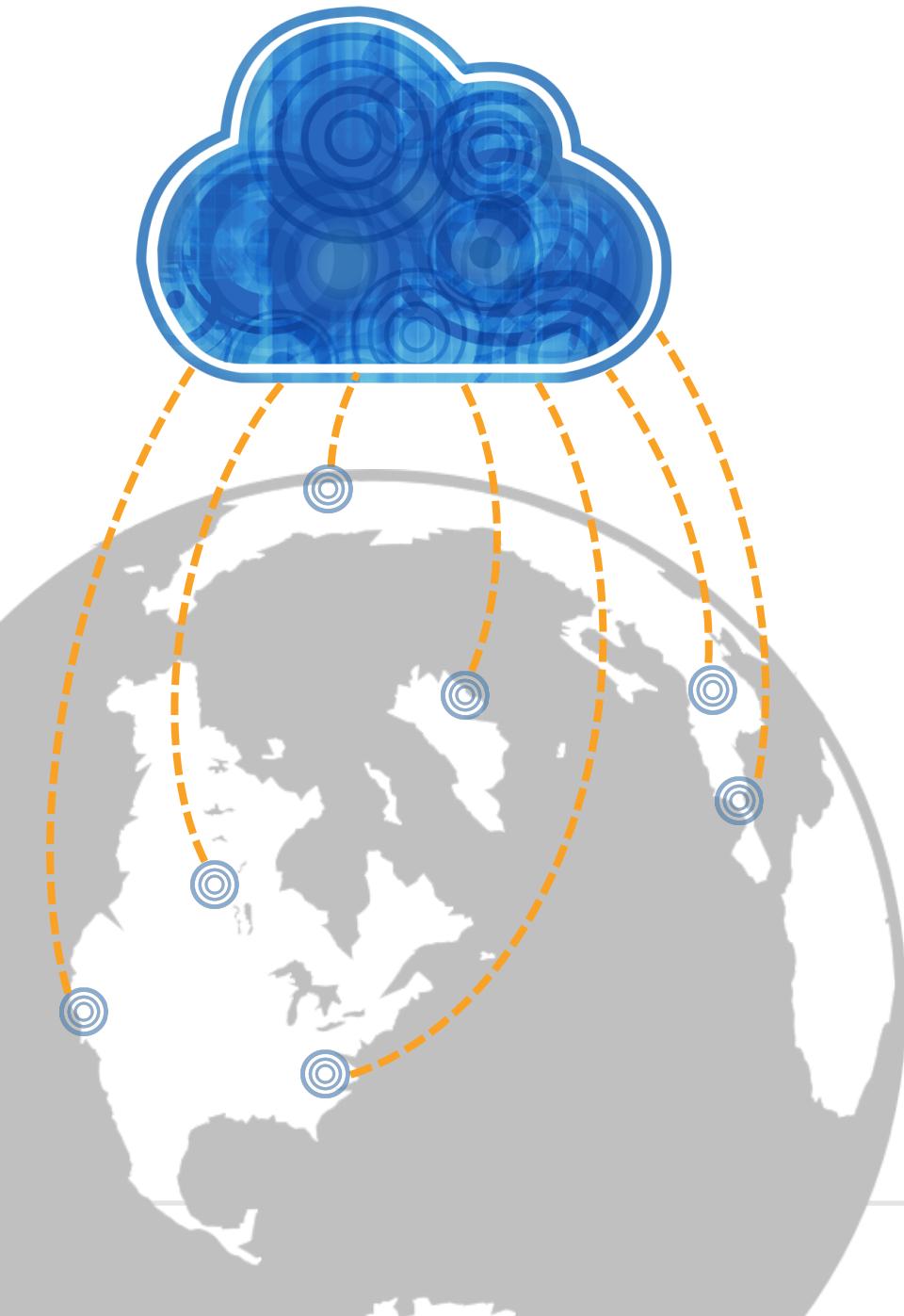


Open source usage is

**EXPLODING**

Yesterday's source  
code are today's

**COMPONENTS**



# APPLICATIONS

are assembled using third party “components,” most of which are open source

In fact, **90%** of a typical application is open source

```
<sourceArchive>struts-core-1.2.8-SONATYPE-001</sourceArchive>  
Presentation Layer
```

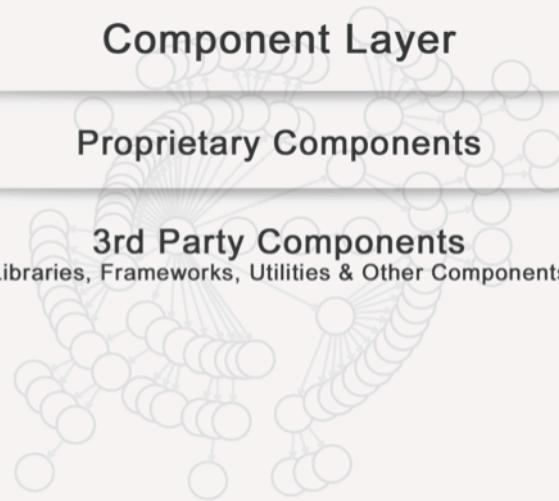
```
<sourceArchive>struts-core-1.2.8-SONATYPE-001</sourceArchive>  
Business Logic
```

## Component Layer

### Proprietary Components

### 3rd Party Components

(Libraries, Frameworks, Utilities & Other Components)



```
01010001010011101010110000101011101010101010010  
01000101010110101000100101010101010101000010101  
Database
```

```
01010001010011101010110000101011101010101010000  
0010101011101010100010101010101010000010101  
Operating System
```

```
01010001010011101010110000101011101010101010000  
0010101011101010100010101010101010000010101  
Firmware
```

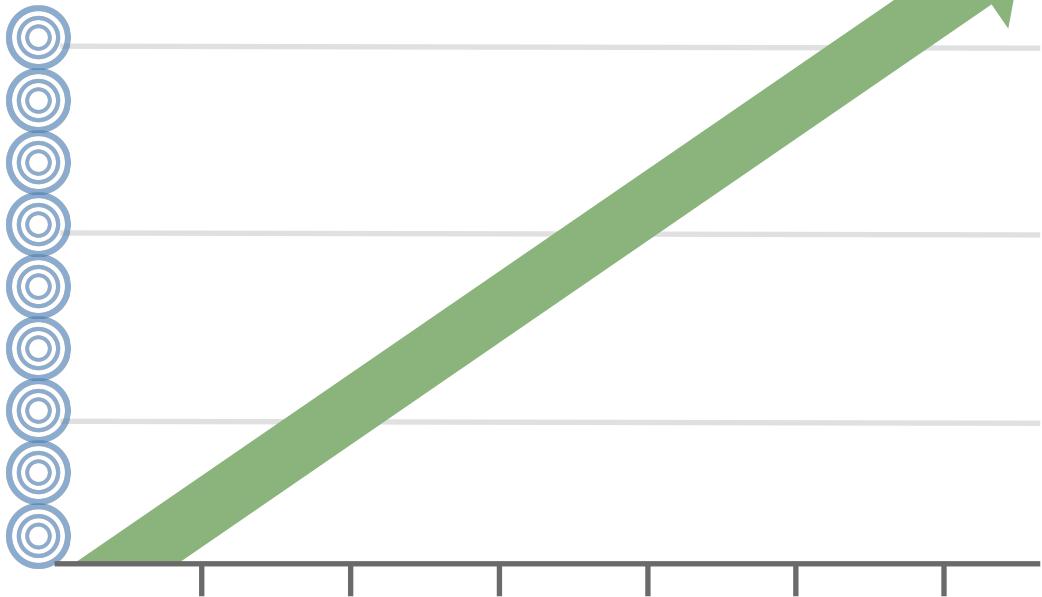
```
01010001010011101010110000101011101010101010000  
00100010101011010100101110101010101000010101  
Network
```

# OPEN SOURCE:

## QUALITY

## INNOVATION

## EFFICIENCY





Modern software development

# **HAS CHANGED**

Our process

# **HASN'T CHANGED ENOUGH**



#### Problem Discovery



#### Network Protection



#### Presentation Layer

ResourceArchive>struts-core-1.2.8-SNAPSHOT.jar  
ResourceScm<https://github.com/sonatype/nexus-public/archive/struts-1.2.8-SONATYPE-001.zip>

#### Business Logic

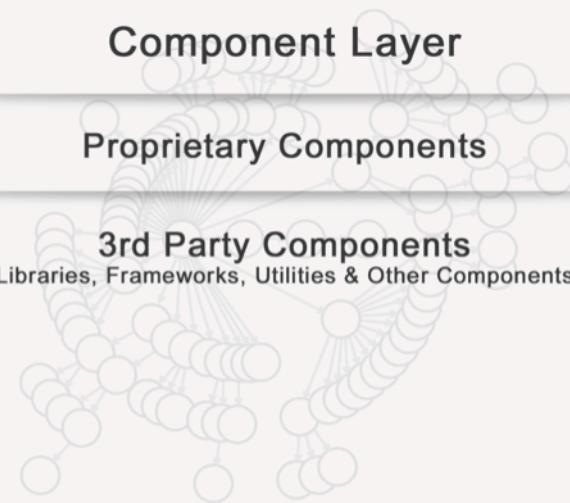
ResourceArchive>struts-core-1.2.8-SNAPSHOT.jar  
ResourceScm<https://github.com/sonatype/nexus-public/archive/struts-1.2.8-SONATYPE-001.zip>

#### Component Layer

##### Proprietary Components

##### 3rd Party Components

(Libraries, Frameworks, Utilities & Other Components)



#### Problem Discovery



#### Network Protection



#### Database

ResourceArchive>struts-core-1.2.8-SNAPSHOT.jar  
ResourceScm<https://github.com/sonatype/nexus-public/archive/struts-1.2.8-SONATYPE-001.zip>

#### Operating System

ResourceArchive>struts-core-1.2.8-SNAPSHOT.jar  
ResourceScm<https://github.com/sonatype/nexus-public/archive/struts-1.2.8-SONATYPE-001.zip>

#### Firmware

ResourceArchive>struts-core-1.2.8-SNAPSHOT.jar  
ResourceScm<https://github.com/sonatype/nexus-public/archive/struts-1.2.8-SONATYPE-001.zip>

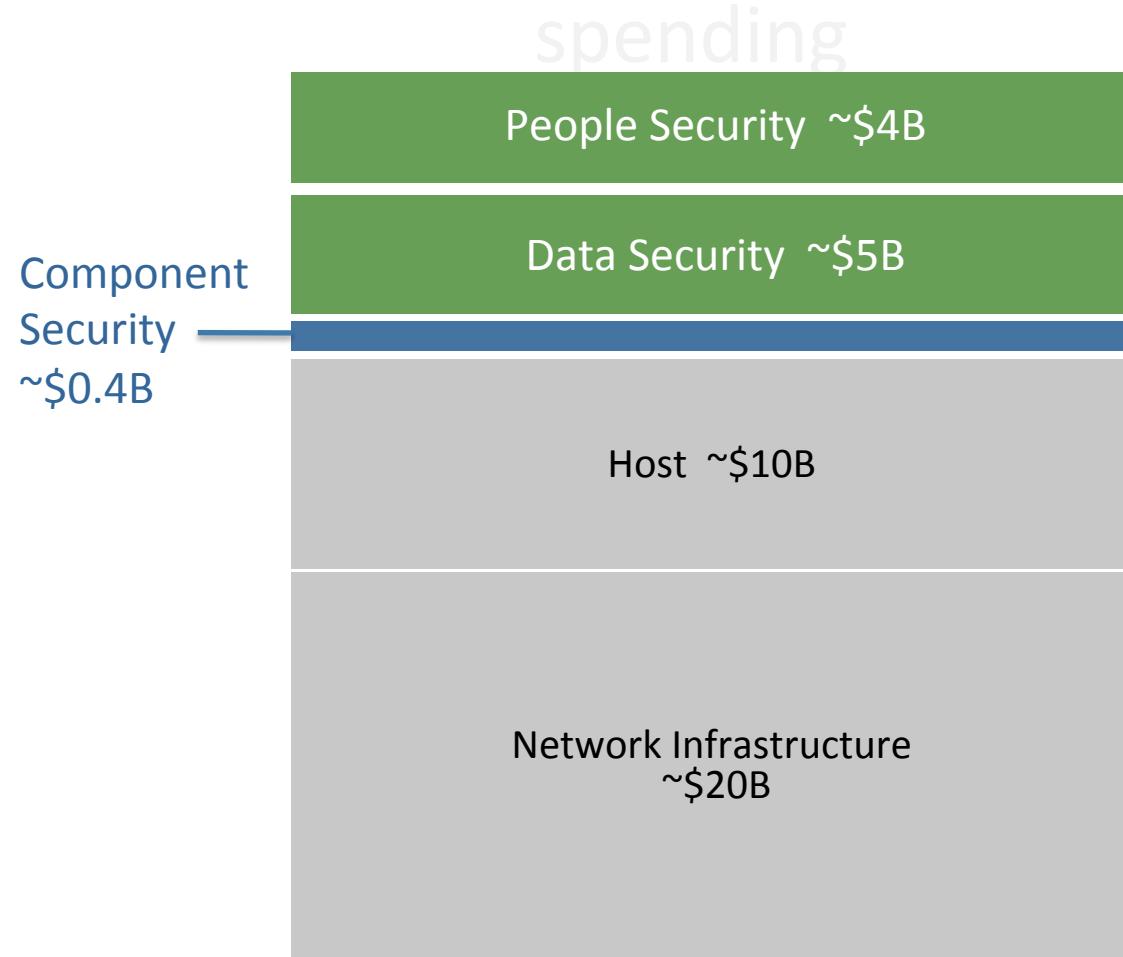
#### Network

ResourceArchive>struts-core-1.2.8-SNAPSHOT.jar  
ResourceScm<https://github.com/sonatype/nexus-public/archive/struts-1.2.8-SONATYPE-001.zip>

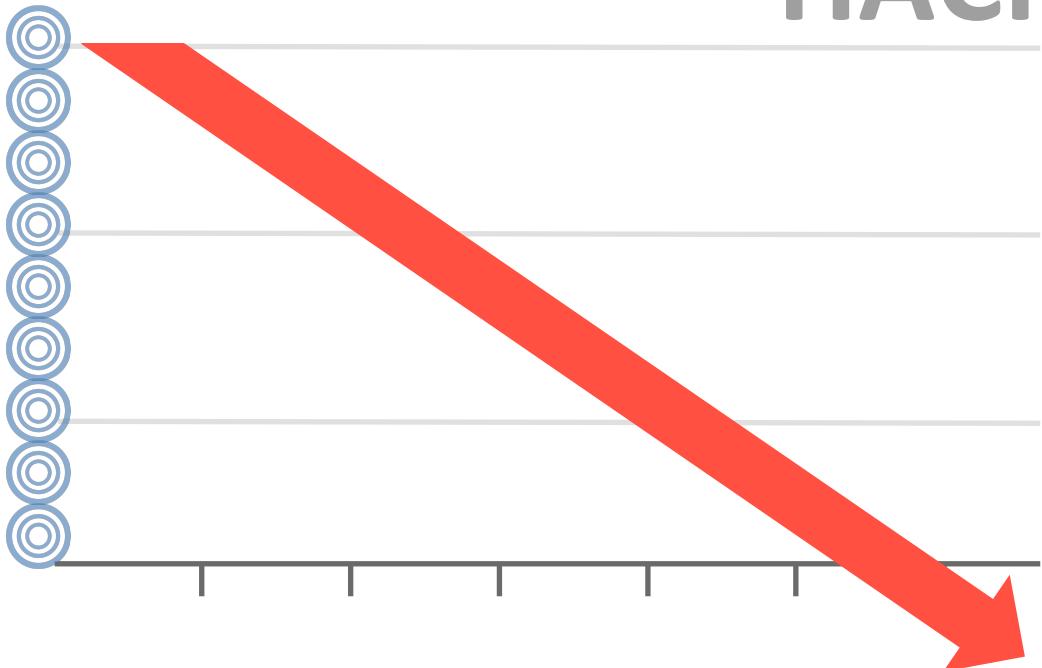


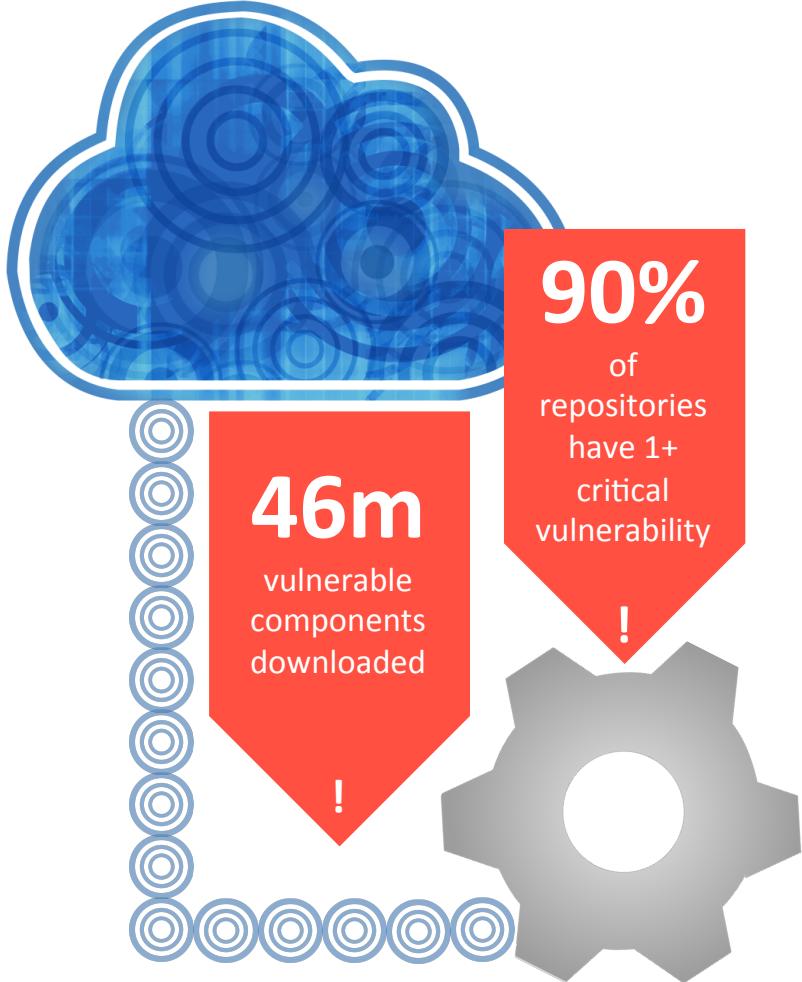
Spending and risk are

# OUT OF SYNC



**NO CONTROLS.**  
**OPEN ACCESS.**  
**HACKER TARGETS.**





Today's approaches

# AREN'T WORKING

**71%**

of apps  
have 1+  
critical or  
severe  
vulnerability

!

COMPONENT  
SELECTION

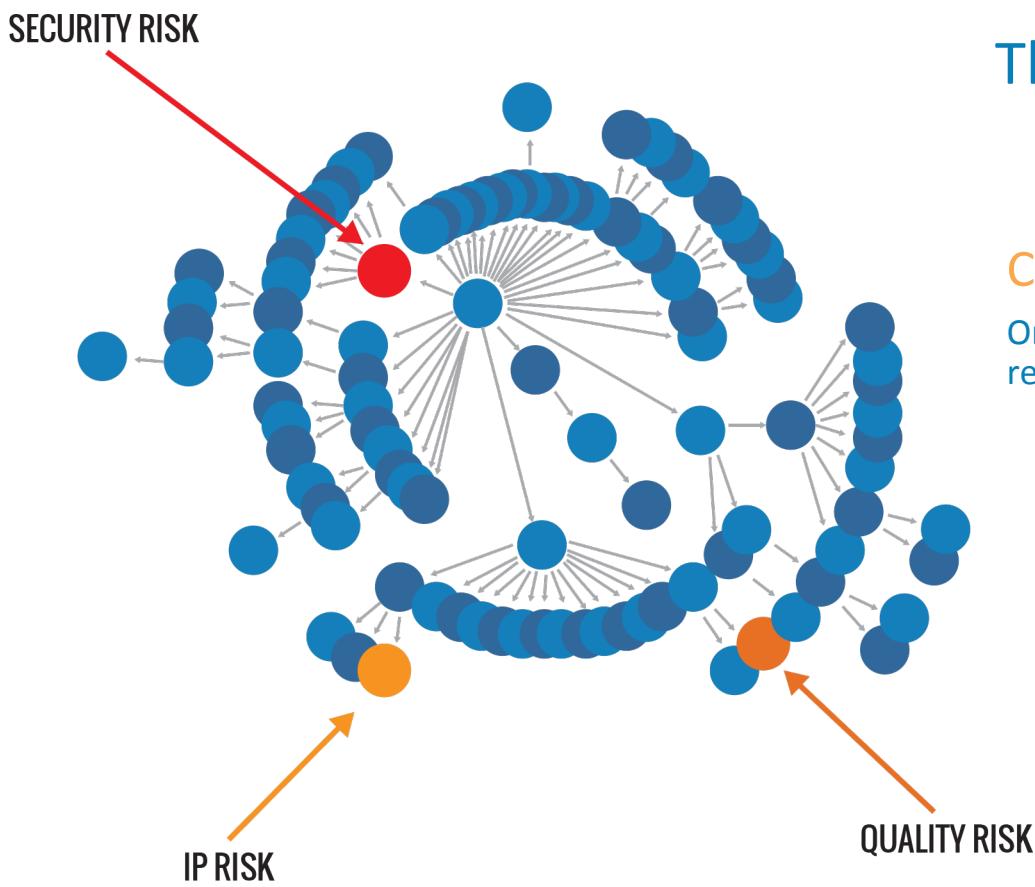
DEVELOPMENT

BUILD AND DEPLOY

PRODUCTION

# WHY?

The problem is too complex  
to manage manually.



## Complexity

One component may  
rely on 100s of others

## Diversity

- 40,000 Projects
- 200M Classes
- 400K Components

## Volume

Typical enterprise consumes  
1,000s of components monthly

## Change

Typical component is  
updated 4X per year

## A MASSIVE SUPPLY CHAIN PROBLEM

**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.



We are not the first  
**INDUSTRY**  
to face this  
**CHALLENGE**

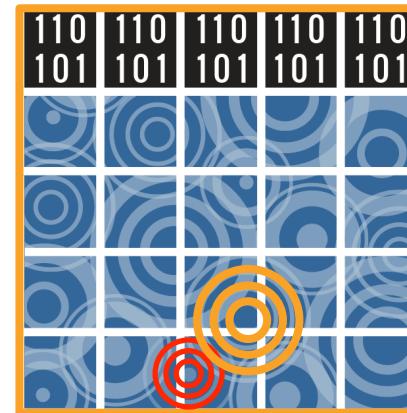
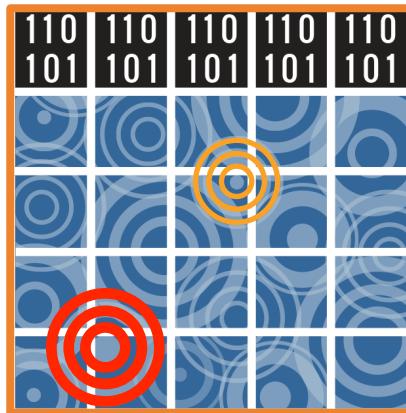
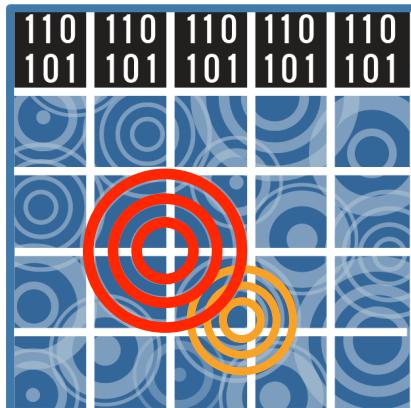




When software was first being written,  
finding exploitable code was like

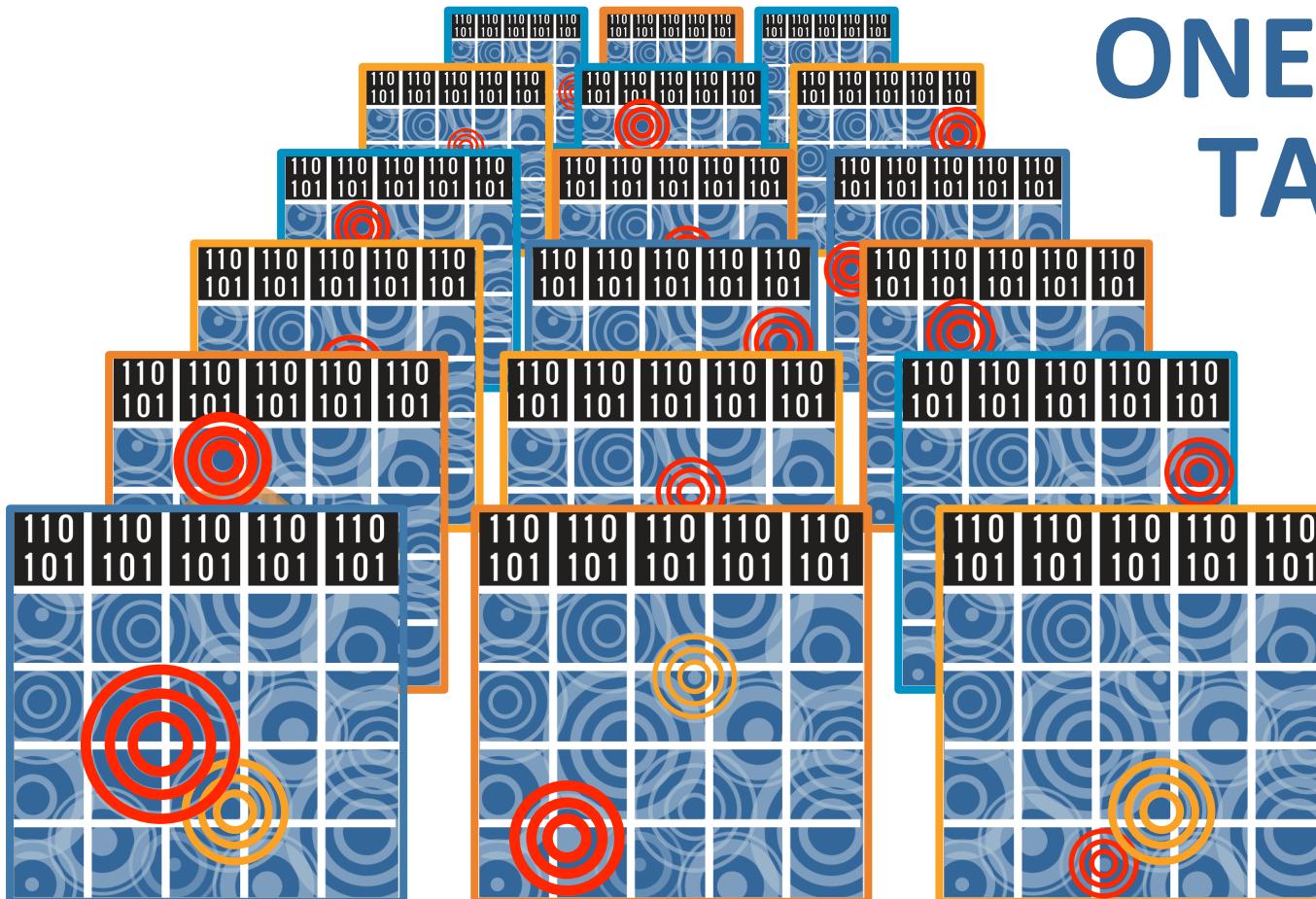
**LOOKING**  
for a needle in a  
**HAYSTACK**

Now that software is  
**ASSEMBLED...**



One risky component,  
multiplied thousands of times:

# ONE EASY TARGET



## FOR EXAMPLE: CVE-2013-2251

Network exploitable

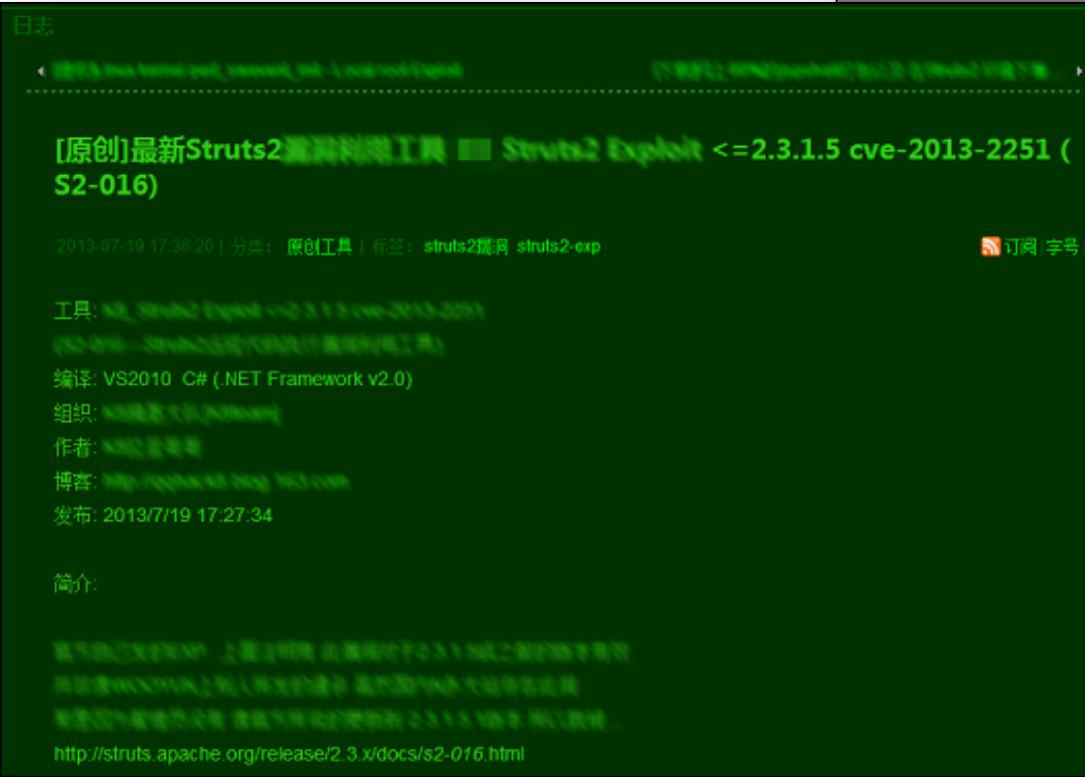
Medium access complexity

No authentication required for exploit

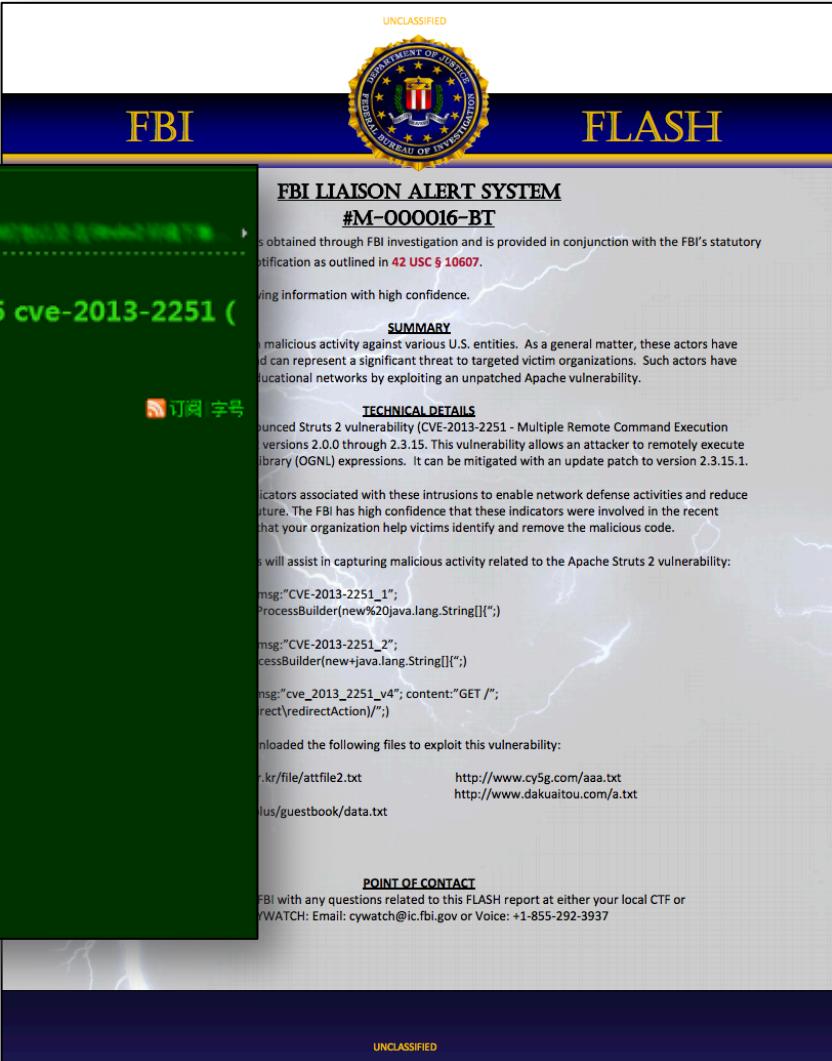
Allows unauthorized disclosure of information;  
allows unauthorized modification; allows disruption of  
service



# WIDESPREAD COMPROMISE



The screenshot shows a blog post titled "[原创]最新Struts2漏洞利用工具 S2-016" dated 2013-07-19. It includes a C# exploit code snippet for CVE-2013-2251, details about the exploit (VS2010, .NET Framework v2.0), and links to the Apache Struts release notes.



**FBI LIAISON ALERT SYSTEM #M-000016-BT**  
This information was obtained through FBI investigation and is provided in conjunction with the FBI's statutory notification as outlined in **42 USC § 10607**.  
**SUMMARY**  
The malicious activity against various U.S. entities. As a general matter, these actors have and can represent a significant threat to targeted victim organizations. Such actors have educational networks by exploiting an unpatched Apache vulnerability.  
**TECHNICAL DETAILS**  
Announced Struts 2 vulnerability (CVE-2013-2251 - Multiple Remote Command Execution versions 2.0.0 through 2.3.15. This vulnerability allows an attacker to remotely execute library (OGNL) expressions. It can be mitigated with an update patch to version 2.3.15.1. Indicators associated with these intrusions to enable network defense activities and reduce future. The FBI has high confidence that these indicators were involved in the recent that your organization help victims identify and remove the malicious code. This will assist in capturing malicious activity related to the Apache Struts 2 vulnerability:  

```
msg:"CVE-2013-2251_1";
ProcessBuilder(new%20java.lang.String[]{});
```

```
msg:"CVE-2013-2251_2";
ProcessBuilder(new+java.lang.String[]{});
```

```
msg:"cve_2013_2251_v4"; content:"GET /";
redirect.sendRedirectAction());
```

Downloaded the following files to exploit this vulnerability:

<a href="#">http://www.cy5g.com/aaa.txt</a>	<a href="#">http://www.dakuitou.com/a.txt</a>
<a href="#">http://www.kr/file/attfile2.txt</a>	<a href="#">http://www.cy5g.com/aaa.txt</a>
<a href="#">http://www.guestbook/data.txt</a>	

**POINT OF CONTACT**  
FBI with any questions related to this FLASH report at either your local CTF or  
CYWATCH: Email: cywatch@ic.fbi.gov or Voice: +1-855-292-3937

### Vulnerability Summary for CVE-2007-6721

**Original release date:** 03/30/2009

Last revised: 01/20/2011

Source: US-CERT/NIST

#### Overview

The Legion of the Bouncy Castle Java Cryptography API before release 1.38 (aka Bouncy Castle 1.38) contains multiple security issues, including a denial of service (DoS) attack vector and remote attack vectors related to "a Bleichenbacher vulnerability in simple RSA padding."

#### Impact

CVSS Severity (version 2.0):

**CVSS v2 Base Score:** 10.0 (HIGH)

**Impact Subscore:** 10.0

**Exploitability Subscore:** 10.0

CVSS Version 2 Metrics:

**Access Vector:** Network exploitable

**Access Complexity:** Low

\*\*NOTE: Access Complexity scored Low due to insufficient information

**Authentication:** Not required to exploit

**Impact Type:** Allows unauthorized disclosure of information; Allows unauthorized modification of data

In 2013, **4,000** organizations downloaded a version of Bouncy Castle with a level 10 vulnerability

**20,000 TIMES ...**

**MORE THAN  
FIVE YEARS**

after the vulnerability was fixed

Vulnerability Summary for CVE-2012-5783

Original release date: 11/04/2012

Last revised: 01/14/2014

Source: US-CERT/NIST

## Overview

Apache Commons HttpClient 3.x, as used in Amazon Flexible Payments, allows for man-in-the-middle attacks to spoof SSL servers via an arbitrary validation check.

## Impact

CVSS Severity (version 2.0):

**CVSS v2 Base Score:** 5.8 (MEDIUM)

**Impact Subscore:** 4.9

**Exploitability Subscore:** 8.6

CVSS Version 2 Metrics:

**Access Vector:** Network exploitable

**Access Complexity:** Medium

**Authentication:** Not required to exploit

**Impact Type:** Allows unauthorized disclosure of information; Allows for unauthorized modification of information

6,916 DIFFERENT

organizations downloaded  
a version of httpclient with  
broken ssl validation (cve-2012-5783)

66,824 TIMES ...

More than **ONE YEAR  
AFTER THE ALERT**

## RISK IN COMPONENTS

- 
- The diagram features a large red circle on the left labeled "RISK IN COMPONENTS". To its right, a light gray circle contains four smaller red circles connected by a curved red line. Each red circle is positioned above a text statement. The top node is labeled "Component usage has **exploded**". The middle-left node is labeled "Applications are the primary vector of **attack**". The middle-right node is labeled "There is a proliferation of **flawed** components". The bottom node is labeled "Current approaches **can't handle** the complexity". A faint "or Yes" is visible near the top left of the gray circle.
- Component usage has **exploded**
  - Applications are the primary vector of **attack**
  - There is a proliferation of **flawed** components
  - Current approaches **can't handle** the complexity

# THOUGHT LEADERS ARE TAKING ACTION

The slide is titled "OWASP Top 10 - 2013 rc1" and "The Ten Most Critical Web Application Security Risks". It features a green header with the OWASP logo and the text "RELEASE CANDIDATE FOR COMMENT". The main content is slide A9, titled "Using Components with Known Vulnerabilities".

**A9 Using Components with Known Vulnerabilities**

Threat Agents	Attack Vectors	Security Weakness	Technical Impacts	Business Impacts	?
?	Exploitability AVERAGE	Prevalence WIDESPREAD	Detectability DIFFICULT	Impact MODERATE	Consider what each vulnerability might mean for the business controlled by the affected application. It could be trivial or it could mean complete compromise.
Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools, expanding the threat agent pool beyond targeted attackers to include chaotic actors.	Attacker identifies a weak component through scanning or manual analysis. They customize the exploit as needed and execute the attack. It gets more difficult if the used component is deep in the application.	Virtually every application has these issues because most development teams don't focus on ensuring their components stay up to date. In many cases, the developers don't even know all the components they are using, never mind their versions. Component dependencies make things even worse.			

**Am I Vulnerable to Known Vulns?**

In theory, it ought to be easy to figure out if you are currently using any vulnerable components or libraries. Unfortunately, vulnerability reports do not always specify exactly which versions of a component are vulnerable in a standard, searchable way. Further, not all libraries use an understandable version numbering system. Worst of all, not all vulnerabilities are reported to a central clearinghouse that is easy to search, although sites like [CVE](#) and [NVD](#) are becoming easier to search.

Determining if you are vulnerable requires searching these databases, as well as keeping abreast of project mailing lists and announcements for anything that might be a vulnerability. If one of your components does have a vulnerability, you should carefully evaluate whether you are actually vulnerable by checking to see if your code uses the part of the component with the vulnerability and whether the flaw could result in an impact you care about.

**How Do I Prevent This?**

One option is not to use components that you didn't write. But realistically, the best way to deal with this risk is to ensure that you keep your components up-to-date. Many open source projects (and other component sources) do not create vulnerability patches for old versions. Instead, most simply fix the problem in the next version.

Software projects should have a process in place to:

- 1) Identify the components and their versions you are using, including all dependencies. (e.g., the [versions](#) plugin).
- 2) Monitor the security of these components in public databases, project mailing lists, and security mailing lists, and keep them up-to-date.
- 3) Establish security policies governing component use, such as requiring certain software development practices, passing security tests, and acceptable licenses.

# HOW NOT TO SOLVE THIS PROBLEM

## ANTI-PATTERNS

Turn off the lights!

## ANTI-PATTERNS

Lock the doors!

## ANTI-PATTERNS

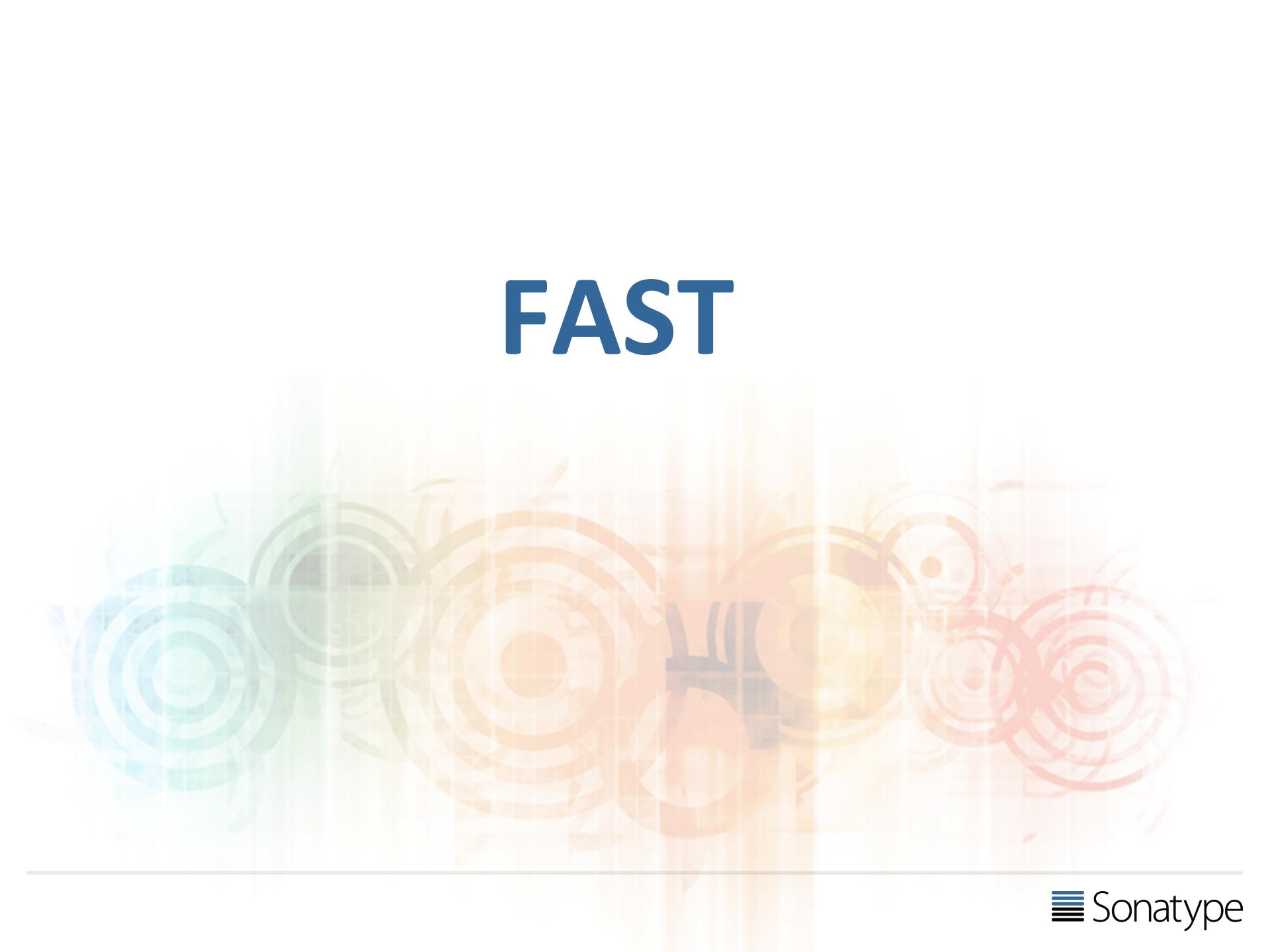
Point fingers!

# THERE IS NO PROBLEM HERE



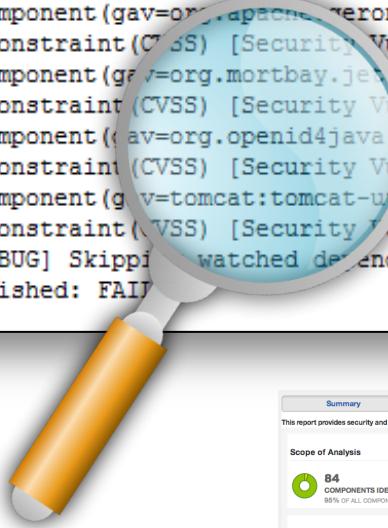
# MODERN SOFTWARE PRACTICES REQUIRE A MODERN APPROACH TO GOVERNANCE

# FAST



# You need to scan hundreds or thousands of applications and their transitive dependencies every day!

```
Policy(Critical Security) [
  Component(gav=commons-httpclient:commons-httpclient:3.1, hash=f0776db1593e215146d2) [
    Constraint(CVSS) [Security Vulnerability Severity >= 5 because: Found 2 Security Vulnerabilities with Severity >= 5] ],
  Component(gav=org.apache.geronimo.framework:geronimo-security:2.1, hash=848d7549ef7ec13ce546) [
    Constraint(CVSS) [Security Vulnerability Severity >= 5 because: Found 6 Security Vulnerabilities with Severity >= 5] ],
  Component(gav=org.mortbay.jetty:jetty:6.1.15, hash=494308fc2d433720c778) [
    Constraint(CVSS) [Security Vulnerability Severity >= 5 because: Found 10 Security Vulnerabilities with Severity >= 5] ],
  Component(gav=org.openid4java:openid4java:0.9.5, hash=62923bdffff066ea088) [
    Constraint(CVSS) [Security Vulnerability Severity >= 5 because: Found 2 Security Vulnerabilities with Severity >= 5] ],
  Component(gav=tomcat:tomcat-util:5.5.23, hash=1249e25aebb15358bedd) [
    Constraint(CVSS) [Security Vulnerability Severity >= 5 because: Found 4 Security Vulnerabilities with Severity >= 5] ]
  [DEBUG] Skipping watched dependency update for build: Sample Project #2768 due to result: FAILURE
Finished: FAILURE
```



Summary Policy Security Issues License Analysis

Threat Level - Problem Components Group Artifact Version Status

Search Level	Search Code	Search Group	Search Artifact	Search Version	Search Status
10	CVN-85999	org.apache.struts.xwork	xwork-core	2.2.1	Open

Component Info Policy Similar Occurrences Licenses Edit Vulnerabilities Labels Audit Log

Override License: Apache-2.0  
Declared License: Apache-2.0  
Observer License: BSD-Apache-2.0  
Group: org.apache.struts.xwork  
Artifact: xwork-core  
Version: 2.2.1  
Highest Security Threat: 10 within 25 security issues  
Categorized: 3 years ago  
Match Ratio: exact  
Identification Source: Sonatype

Popularity License Conflict License Risk Security Alerts

Over The Version New

Dependency Depth

1	2	3	4	5
10	10	10	10	10

1	2	3	4	5
10	10	10	10	10

# AUTOMATE

1. Humans define policy
2. Machines automate the implementation of policy
3. Humans manage exceptions

# PRECISE



## BE SPECIFIC

# No Noise!



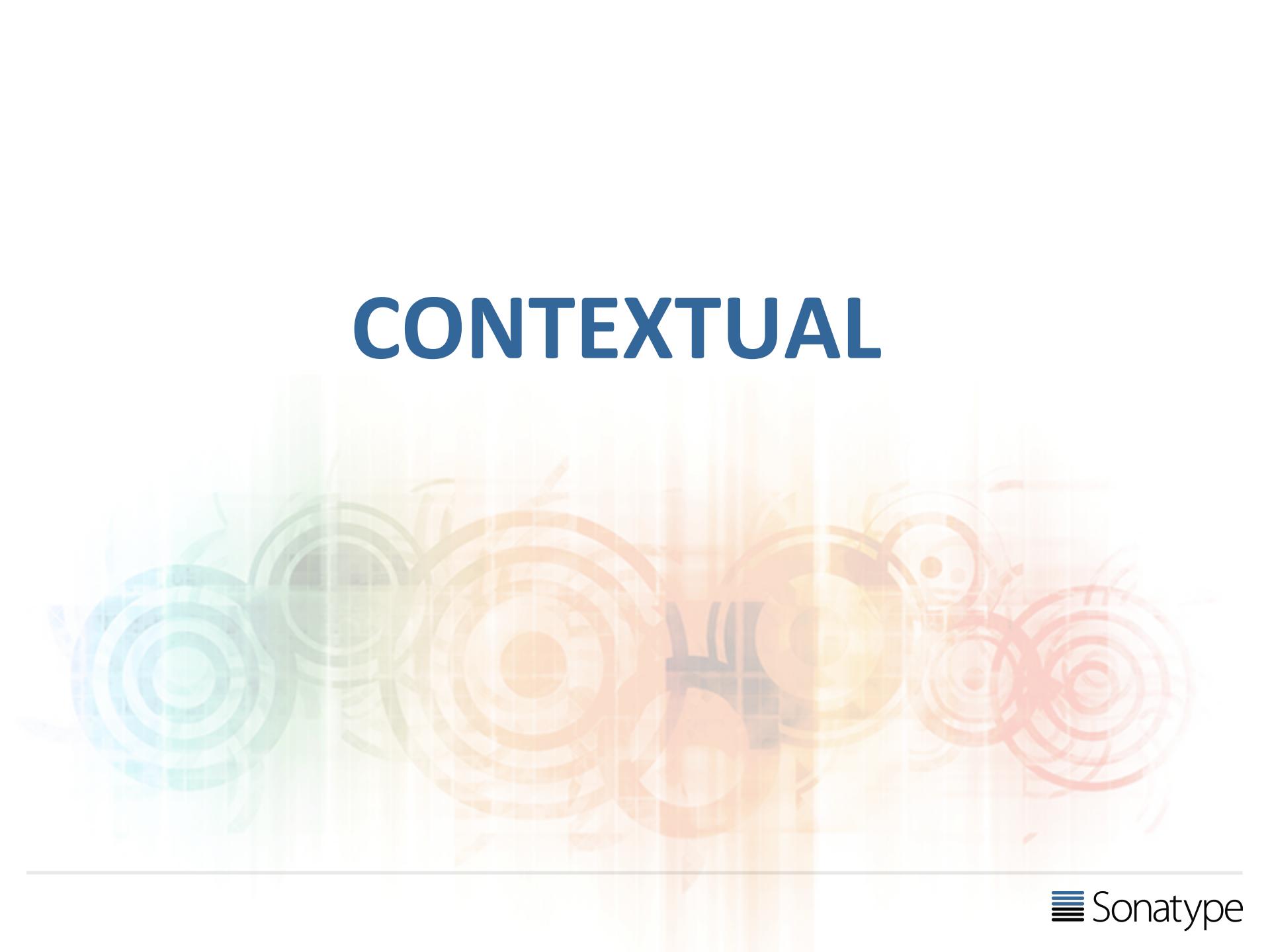
- There is a world of difference between saying "Struts is approved" and saying "Struts 2.3.15.4 is **good** and ~~Struts 2.3.15.0 ANY OLDER VERSION~~ will **get your system owned**"
- Reduce the Noise Level - Provide only verified accurate information to your development teams

Scan found 50,313 “issues” →



Real issue count: 204 →

# CONTEXTUAL



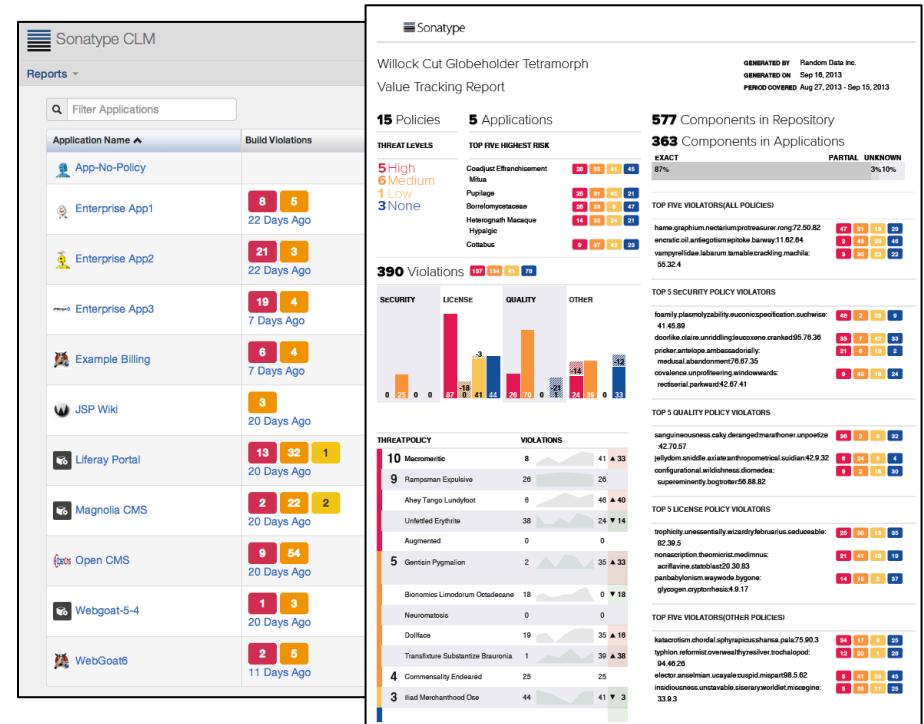
# WHY CONTEXT MATTERS

- SQL Injection vulnerabilities don't affect applications without databases.
- CopyLeft may not be a problem for internal applications or services.
- I need information that applies to my application.

# CONTEXTUAL

Consume information and apply policy in the context of your applications, organizations and enterprise via hierachal policy and reporting

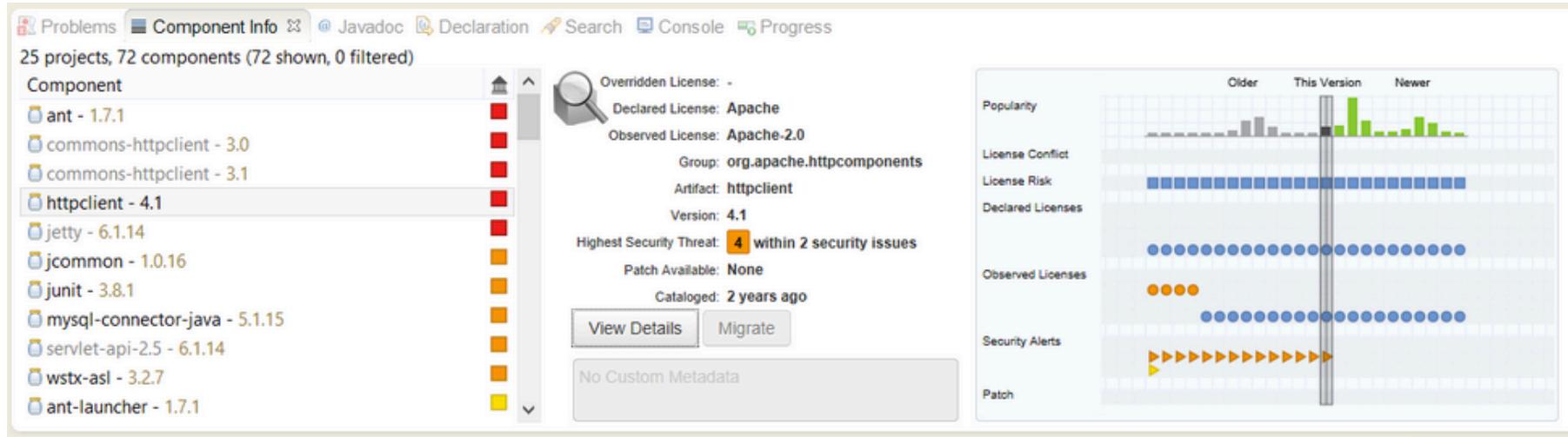
One Size does Not Fit All!



# ACTIONABLE



## POLICIES ENSURE DEVELOPERS START WITH RIGHT COMPONENTS



Comprehensive  
intelligence allows  
developer to start with the  
best component

"I can quickly pick the best component from the start, eliminating downstream rework."  
*Lead Developer*

## PROVIDE A SOLUTION

- Now that you've told me about a problem, tell me what I can do to fix it.
- Suggest alternatives.
- Even if I don't completely understand the risk,  
**if you show me an easy fix, I will take it.**

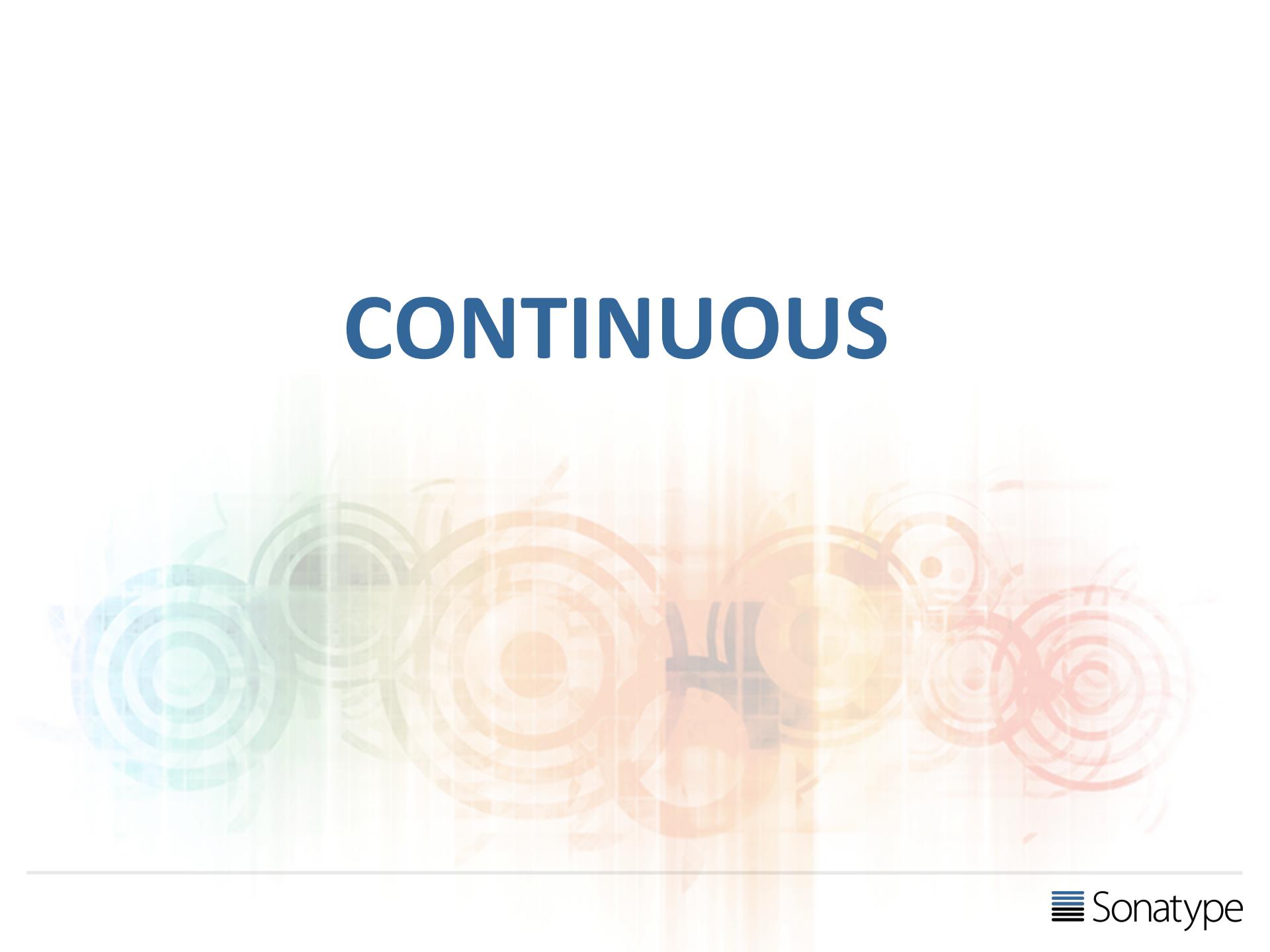
# EASY TO CONSUME

Provide stakeholders actionable, easy to consume information to remediate problems

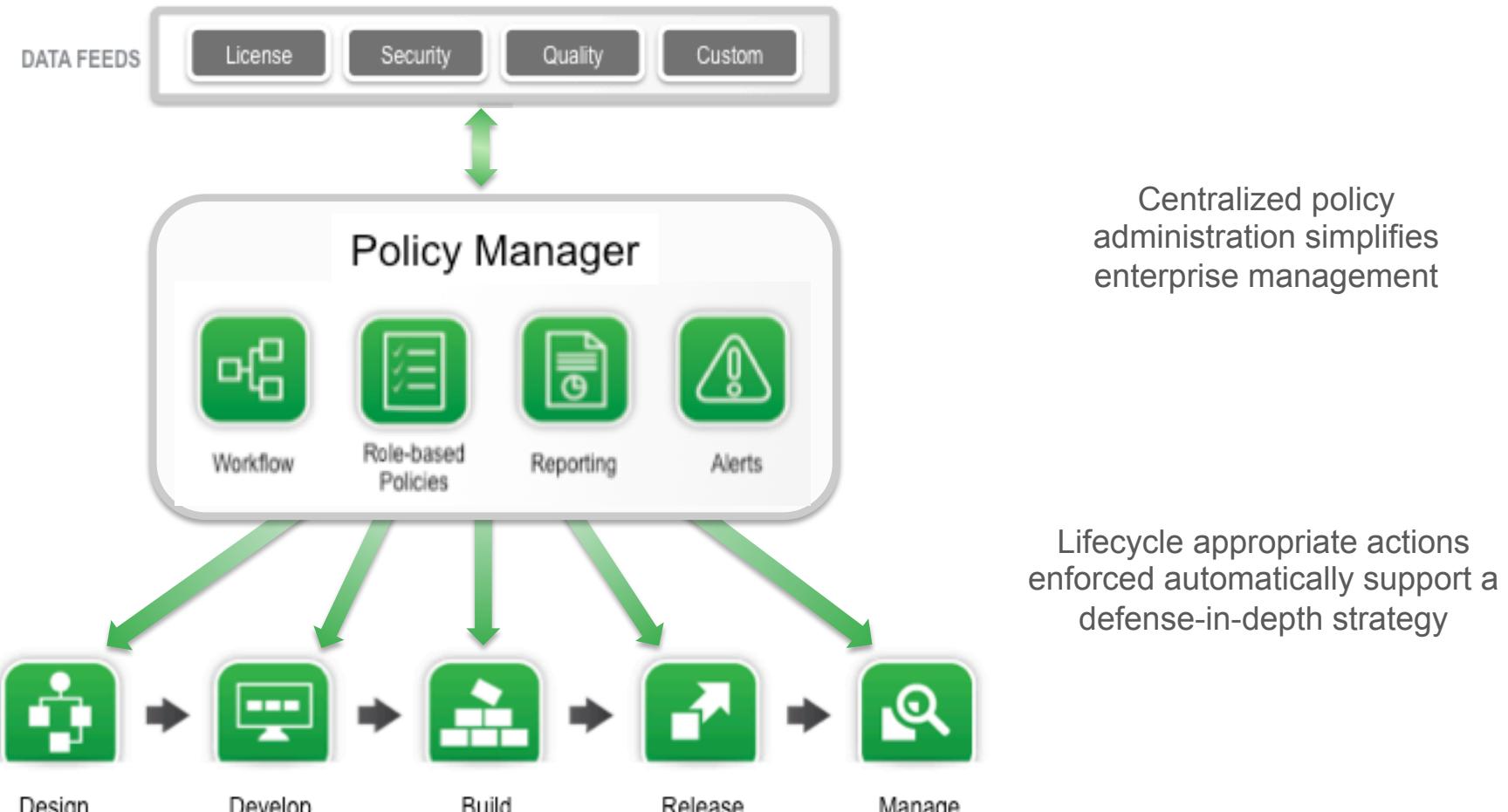
The screenshot displays three main panels illustrating the Sonatype IQ tool's functionality:

- Left Panel (Dependency Management):** Shows the Maven interface with a tree view of dependencies, a central pom.xml editor, and a Problems tab listing 72 components across 25 projects.
- Middle Panel (Dependency Security Overview):** An "Overview" section shows 16 violations (5 New, 3 Updated, 8 Muted) and 1,543 applications. A detailed view for "asm : asm-tree : 2.2.3" highlights a HIGH threat level (9 alerts) and provides a "Compare" feature to analyze alternative versions like guava, jmdns, javax.mail, and javax.jndi.
- Right Panel (Dependency Security Analysis):** A detailed view for "asm : asm-tree : 2.2.3" shows 8 events, 7 violations, and 24 applications. It includes a "5 Month View" chart tracking violations and remediations over time, and an "Events" log with entries such as "Data Scraper Stopped" and "Violation Detected".

# CONTINUOUS



## POLICIES PROVIDE FOUNDATION FOR GOVERNANCE



“Just by using CLM we are enforcing policy.”

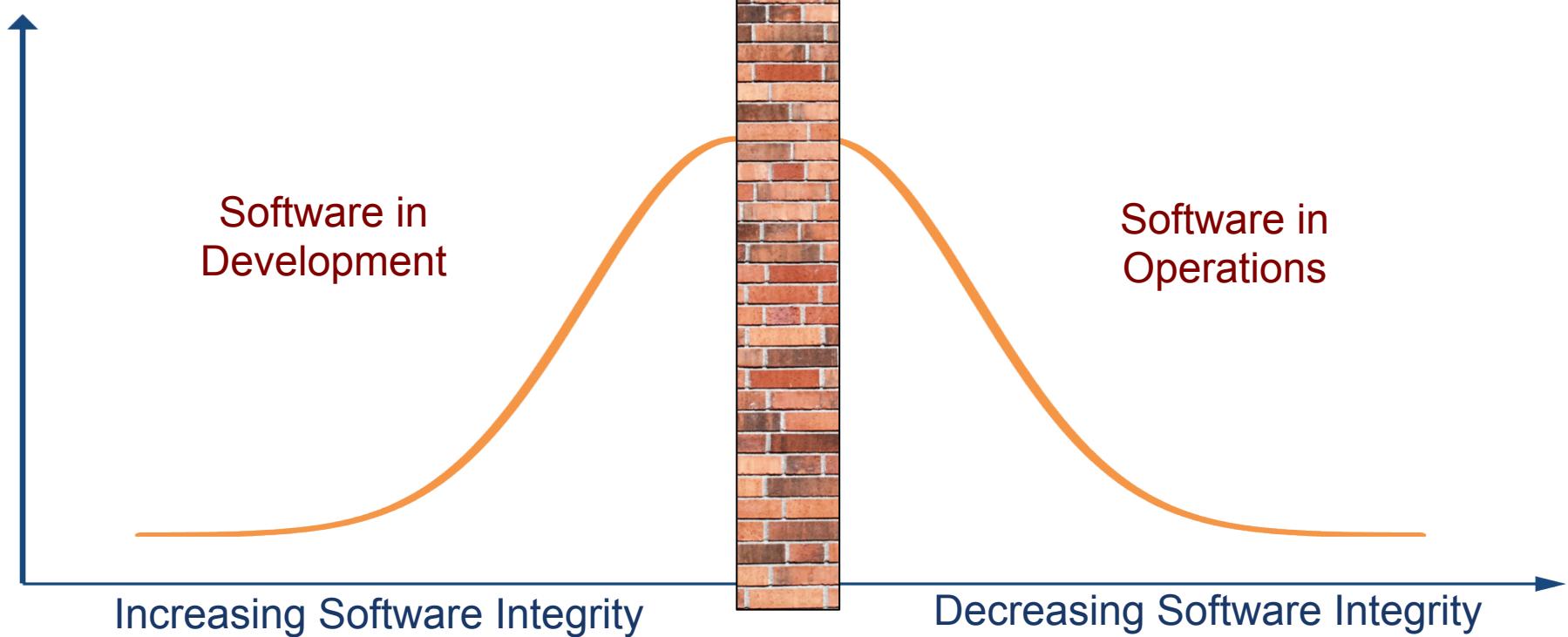
CISO



Applications don't age like wine,

**THEY AGE  
LIKE MILK**

## UNMAINTAINED APPLICATIONS BECOME UNMAINTAINABLE



“It is an inconvenient reality that software integrity degrades the moment development ends.”  
*Industry Analyst*

# **SYSTEM.EXIT(0);**

Q& A



We make it **EASY** to create  
**TRUSTED APPLICATIONS**  
and keep them that way  
**OVER TIME**