

# Application Vulnerability Management

- Application security teams use automated static and dynamic test results as well as manual testing results to assess the security of an application
- Each test delivers results in different formats
- Different test platforms describe same flaws differently, creating duplicates
- Security teams end up using spreadsheets to keep track manually
- It is extremely difficult to prioritize the severity of flaws as a result
- Software development teams receive unmanageable reports and only a small portion of the flaws get fixed

# The Result

- Application vulnerabilities persist in applications:
  - \*\*Average serious vulnerabilities found per website per year is 79
  - \*\*Average days website exposed to one serious vulnerability is 231 days
  - \*\*Overall percentage of serious vulnerabilities that are fixed annually is only 63%
- Part of that problem is there is no easy way for the security team and application development teams to work together on these issues
- Remediation quickly becomes an overwhelming project
- Trending reports that track the number of reduced vulnerabilities are impossible to create

**\*\*WhiteHat Statistics Report (Summer 2012):**  
[https://www.whitehatsec.com/assets/WPstats\\_summer12\\_12th.pdf](https://www.whitehatsec.com/assets/WPstats_summer12_12th.pdf)

## Vulnerability Fun Facts:

Industry	Annual Avg. Vulnerabilities	Avg. Time-to-Fix (Days)	Average Remediation	Window of Exposure (Days)
ALL	79	38	63%	231
Banking	17	45	74%	185
Education	53	30	46%	261
Financial Services	67	80	63%	227
Healthcare	48	35	63%	239
Insurance	92	40	58%	211
IT	85	35	57%	208
Manufacturing	30	17	50%	252
Retail	121	27	66%	238
Social Networking	31	41	62%	264
Telecom	52	50	69%	271
Non-Profit	37	94	56%	320
Energy	31	4	40%	250

- Average number of serious vulnerabilities found per website per year is 79 \*\*
- Serious Vulnerabilities were fixed in ~38 days \*\*
- Percentage of serious vulnerabilities fixed annually is only 63% \*\*
- Average number of days a website is exposed, at least one serious vulnerability ~231 days

WhiteHat Statistics Report (Summer 2012):

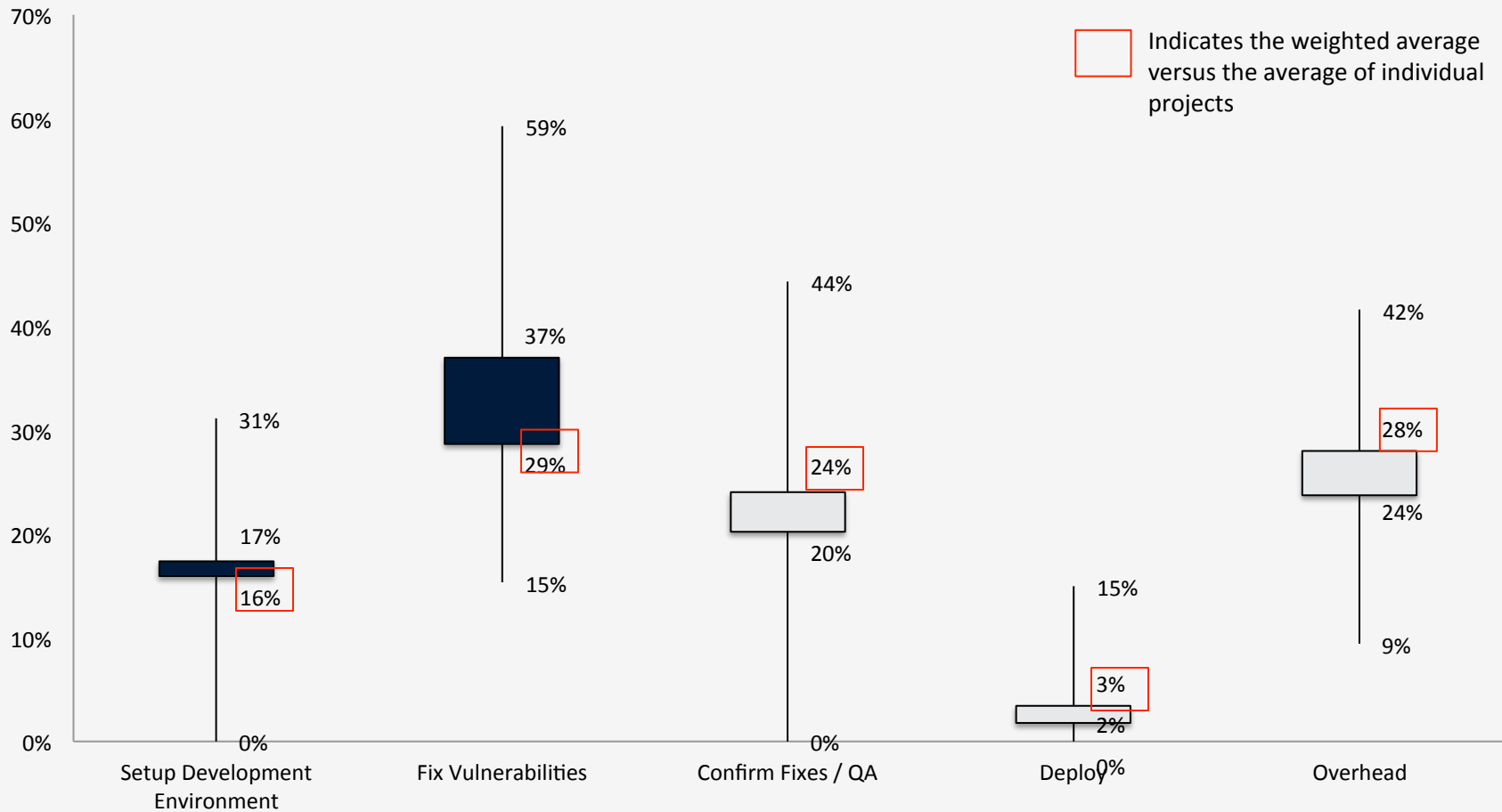
[https://www.whitehatsec.com/assets/WPstats\\_summer12\\_12th.pdf](https://www.whitehatsec.com/assets/WPstats_summer12_12th.pdf)

## Vulnerability Remediation Data

Vulnerability Type	Sample Count	Average Fix (minutes)
Dead Code (unused methods)	465	2.6
Poor logging: system output stream	83	2.9
Poor Error Handling: Empty catch block	180	6.8
Lack of Authorization check	61	6.9
Unsafe threading	301	8.5
ASP.NET non-serializable object in session	42	9.3
XSS (stored)	1023	9.6
Null Dereference	157	10.2
Missing Null Check	46	15.7
XSS (reflected)	25	16.2
Redundant null check	21	17.1
SQL injection	30	97.5



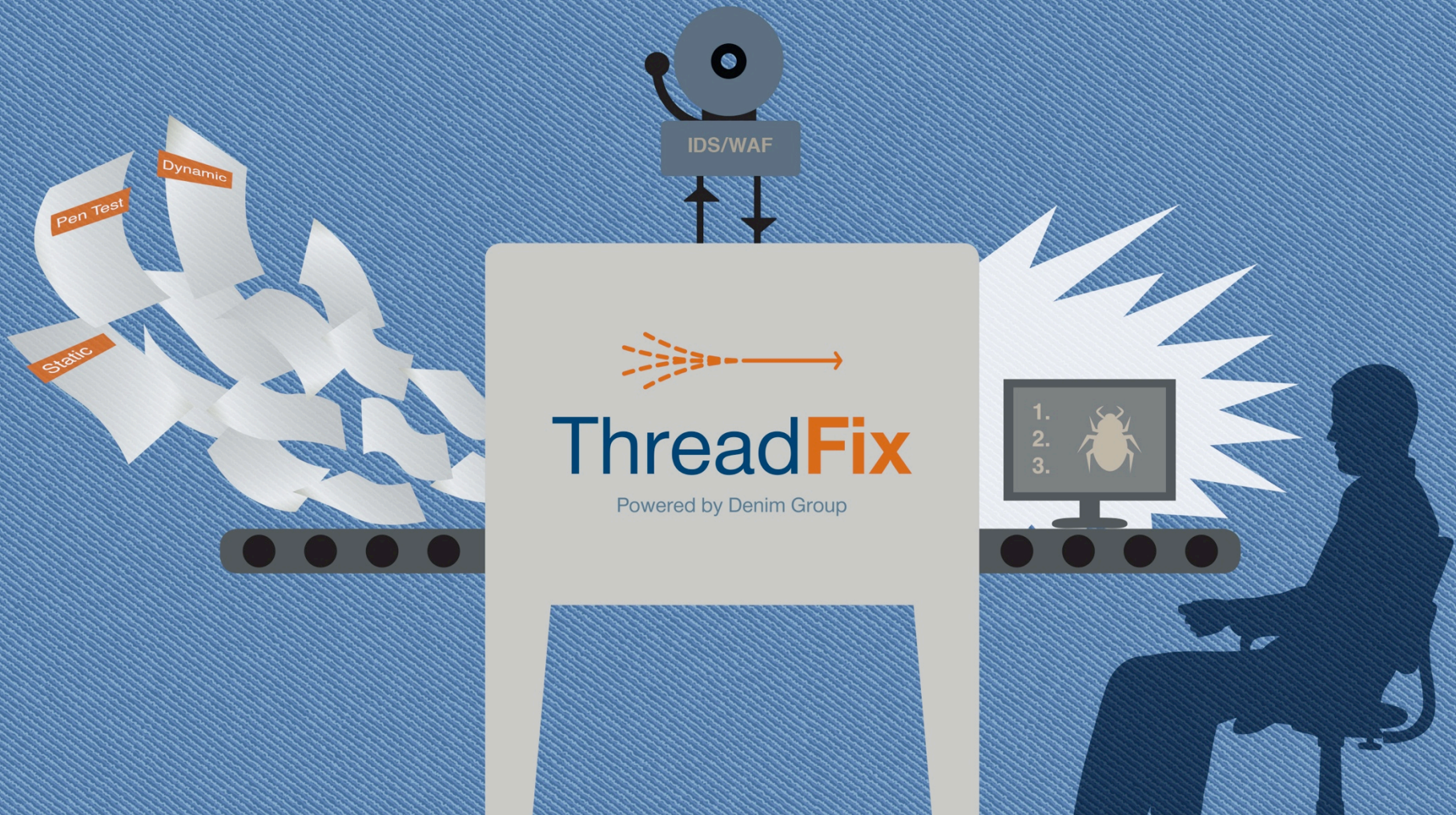
# Where Is Time Being Spent?



# ThreadFix

Accelerate Software Remediation

ThreadFix is a software vulnerability aggregation and management system that helps organizations aggregate vulnerability data, generate virtual patches, and interact with software defect tracking systems.





- **Open source vulnerability management and aggregation platform:**
  - Allows software security teams to reduce the time to remediate software vulnerabilities
  - Enables managers to speak intelligently about the status / trends of software security within their organization.
- **Features/Benefits:**
  - Imports dynamic, static and manual testing results into a centralized platform
  - Removes duplicate findings across testing platforms to provide a prioritized list of security faults
  - Eases communication across development, security and QA teams
  - Exports prioritized list into defect tracker of choice to streamline software remediation efforts
  - Auto generates web application firewall rules to protect data during vulnerability remediation
  - Empowers managers with vulnerability trending reports to pinpoint issues and illustrate application security progress
  - Benchmark security practice improvement against industry standards
- Freely available under the Mozilla Public License (MPL) 2.0
- Download available at: [www.denimgroup.com/threadfix](http://www.denimgroup.com/threadfix)



# List of Supported Tools / Technologies:

## Dynamic Scanners

*Acunetix*  
*Arachni*  
*Burp Suite*  
*HP WebInspect*  
*IBM Security AppScan Standard*  
*IBM Security AppScan Enterprise*  
*Mavituna Security Netsparker*  
*NTO Spider*  
*OWASP Zed Attack Proxy*  
*Tenable Nessus*  
*Skipfish*  
*w3aF*

## Static Scanners

*FindBugs*  
*IBM Security AppScan Source*  
*HP Fortify SCA*  
*Microsoft CAT.NET*  
*Brakeman*

## SaaS Testing Platforms

*WhiteHat*  
*Veracode*  
*QualysGuard WAS*

## IDS/IPS and WAF

*DenyAll*  
*F5*  
*Imperva*  
*Mod\_Security*  
*Snort*

## Defect Trackers

*Atlassian JIRA*  
*Microsoft Team Foundation Server*  
*Mozilla Bugzilla*

## Known Vulnerable Component Scanner

*Dependency Check*



## Large Range of Tool Compatibility

VERACODE



IBM

PortSwigger  
web security



iMPERVA

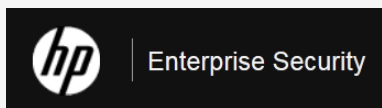
ATLASSIAN  
JIRA



skipfish



acunetix



# What is a Unique Vulnerability?

- (CWE, Relative URL)
  - Predictable resource location
  - Directory listing misconfiguration
- (CWE, Relative URL, Injection Point)
  - SQL injection
  - Cross-site Scripting (XSS)
- Injection points
  - Parameters – GET/POST
  - Cookies
  - Other headers

# Why Common Weakness Enumeration (CWE)?

- Every tool has their own “spin” on naming vulnerabilities
- OWASP Top 10 / WASC 24 are helpful but not comprehensive
- CWE is exhaustive (though a bit sprawling at times)
- Reasonably well-adopted standard
- Many tools have mappings to CWE for their results
- Main site: <http://cwe.mitre.org/>

# What Can We Do With ThreadFix?

- Create a consolidated view of your applications and vulnerabilities
- Prioritize application risk decisions based on data
- Translate vulnerabilities to developers in the tools they are already using

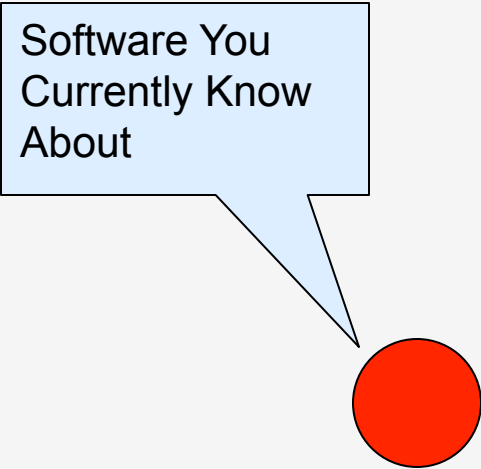




# Create a consolidated view of your applications and vulnerabilities

# What Is Your Software Attack Surface?

Software You  
Currently Know  
About



What?

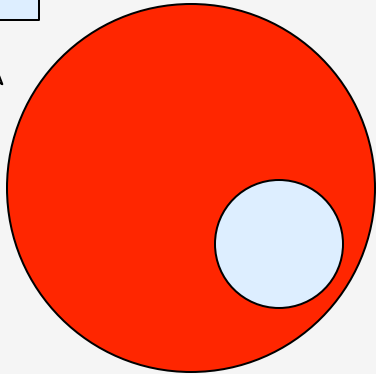
- Critical legacy systems
- Notable web applications

Why?

- Lots of value flows through it
- Auditors hassle you about it
- Formal SLAs with customers mention it
- Bad guys found it and caused an incident (oops)

# What Is Your Software Attack Surface?

Add In the Rest  
of the Web  
Applications You  
Actually Develop  
and Maintain



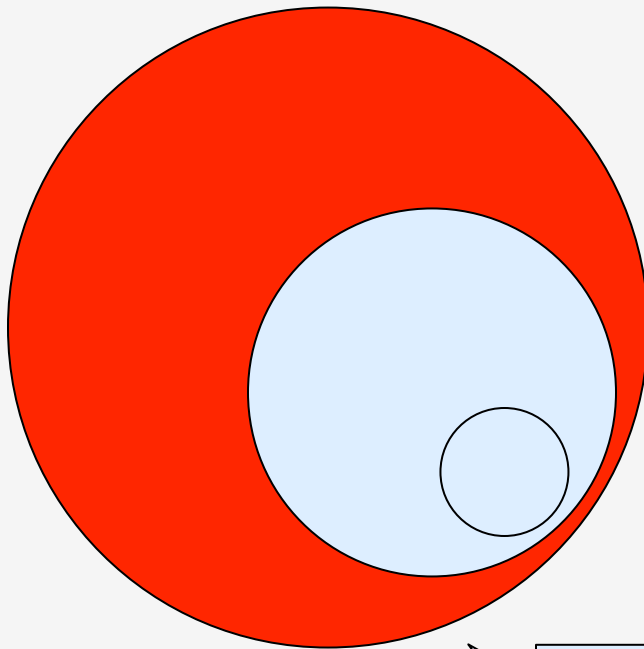
## What?

- Line of business applications
- Event-specific applications

## Why Did You Miss Them?

- Forgot it was there
- Line of business procured through non-standard channels
- Picked it up through a merger / acquisition

# What Is Your Software Attack Surface?



Add In the  
Software You  
Bought from  
Somewhere

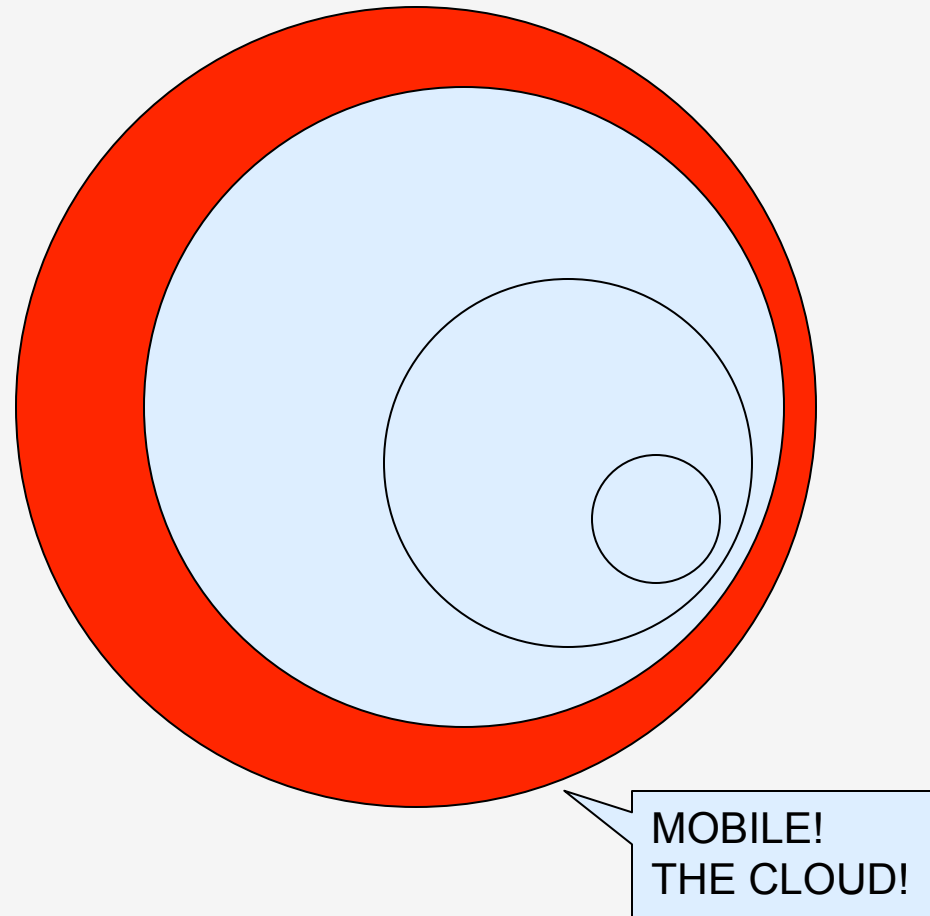
## What?

- More line of business applications
- Support applications
- Infrastructure applications

## Why Did You Miss Them?

- Most scanner only really work on web applications so no vendors pester you about your non-web applications
- Assume the application vendor is handling security

# What Is Your Software Attack Surface?



## What?

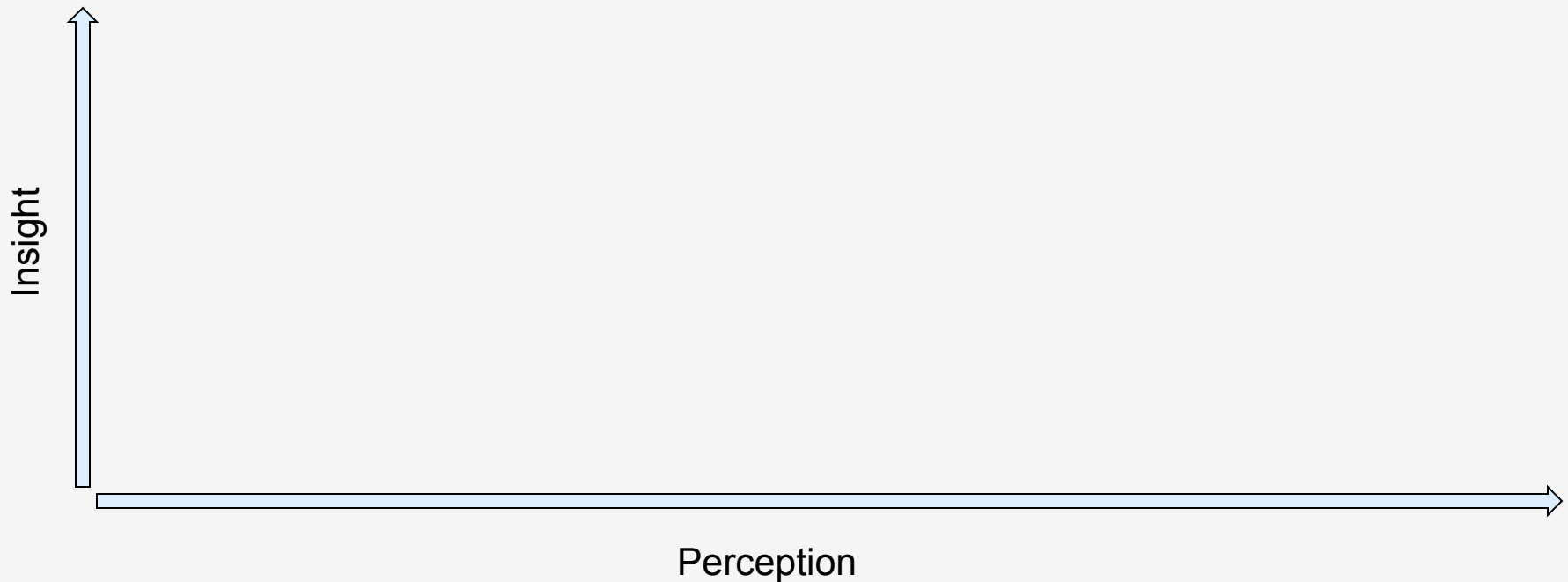
- Support for line of business functions
- Marketing and promotion

## Why Did You Miss Them?

- Any jerk with a credit card and the ability to submit an expense report is now runs their own private procurement office

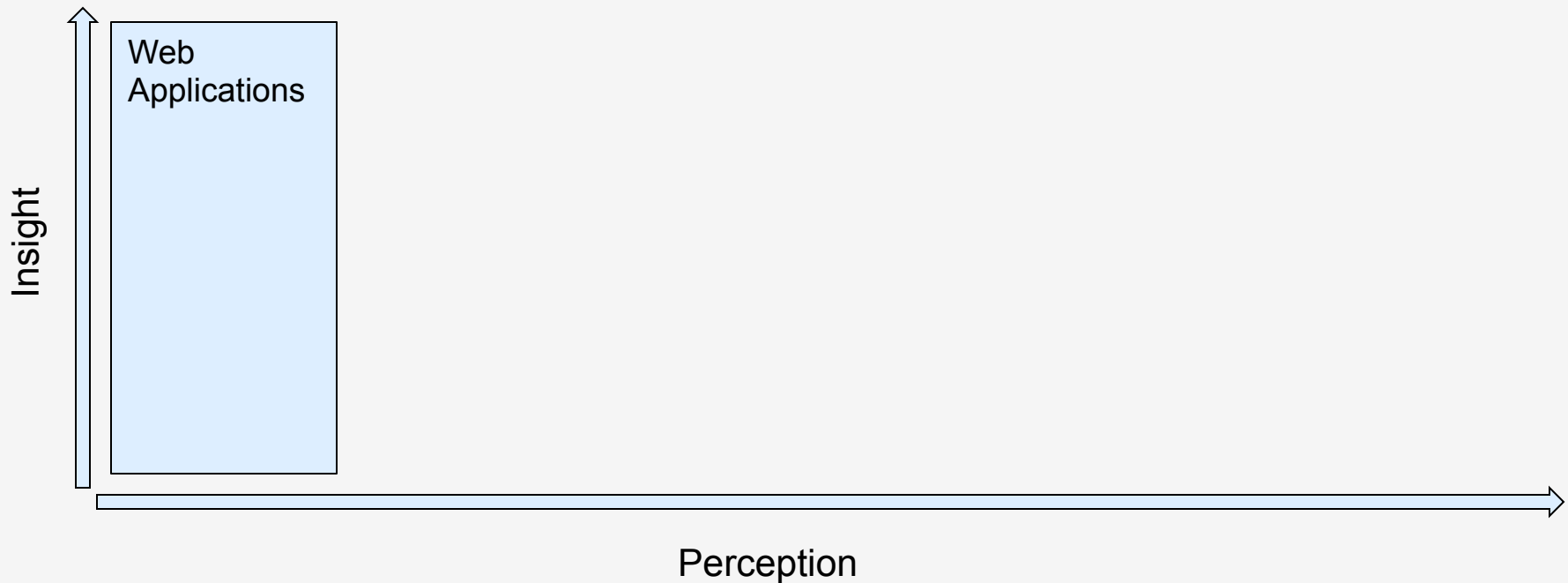
# Attack Surface: The Security Officer's Journey

- Two Dimensions:
  - Perception of Software Attack Surface
  - Insight into Exposed Assets



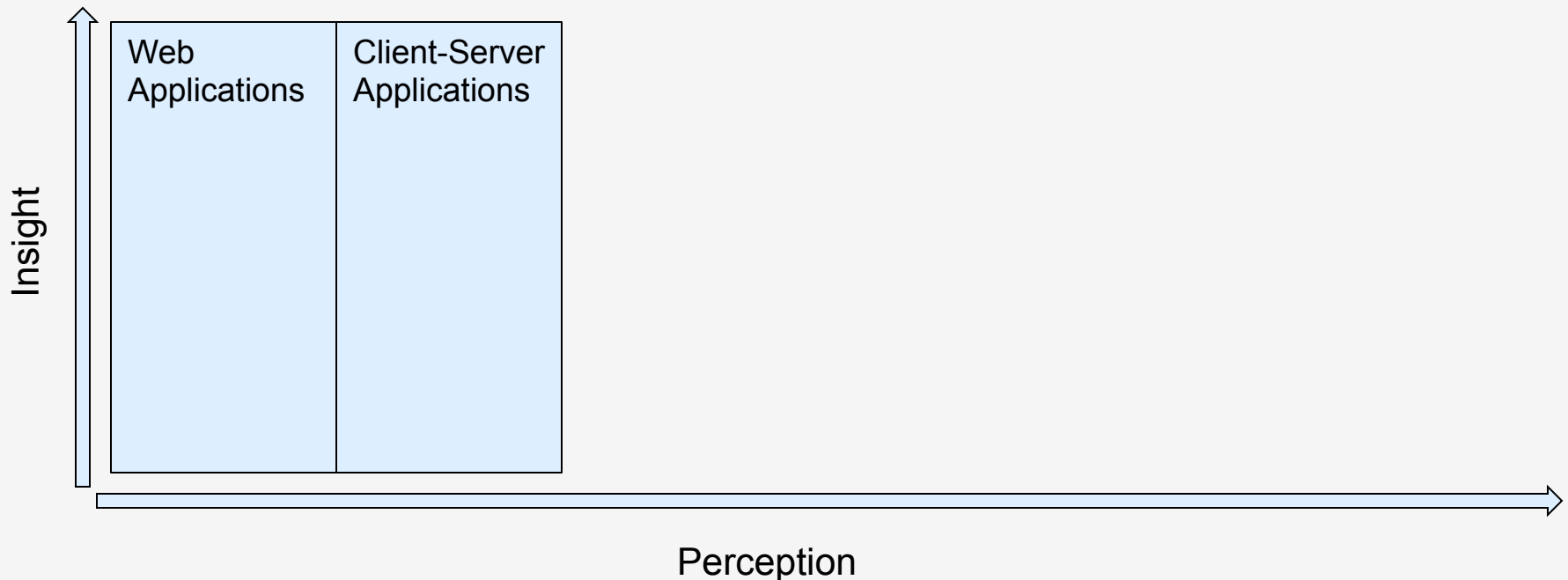
# Attack Surface: The Security Officer's Journey

- As perception of the problem of attack surface widens the scope of the problem increases



# Attack Surface: The Security Officer's Journey

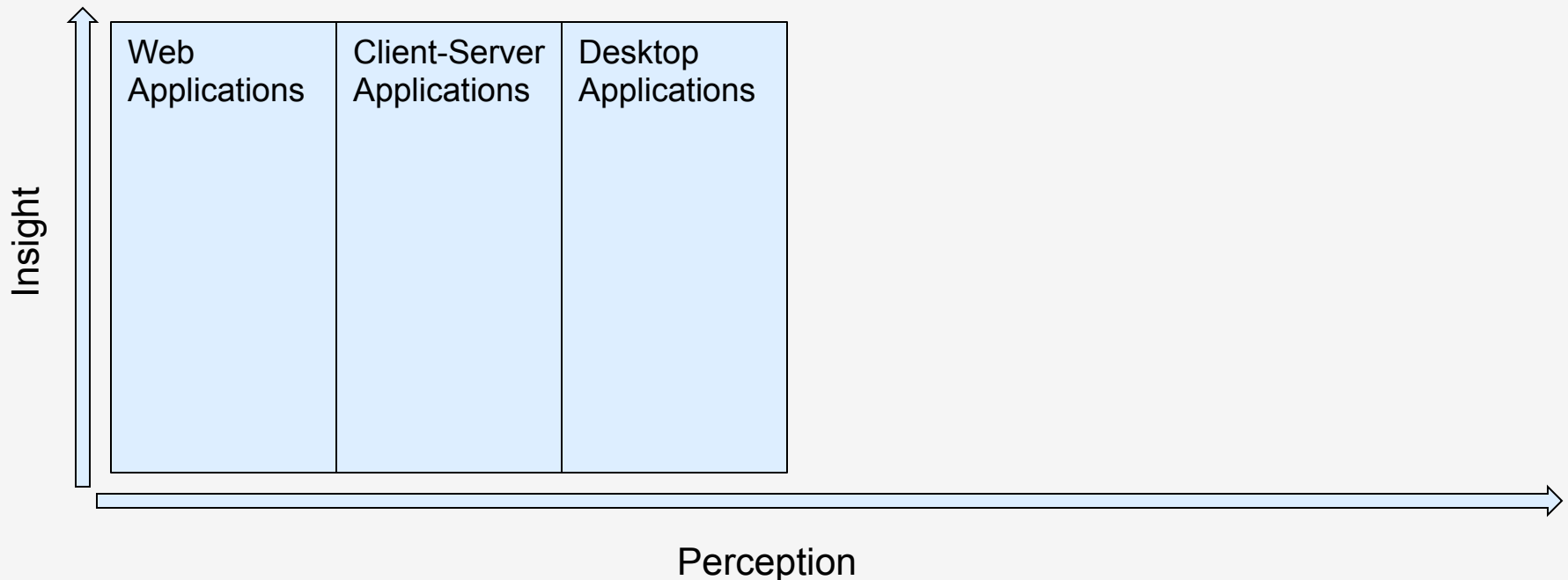
- As perception of the problem of attack surface widens the scope of the problem increases





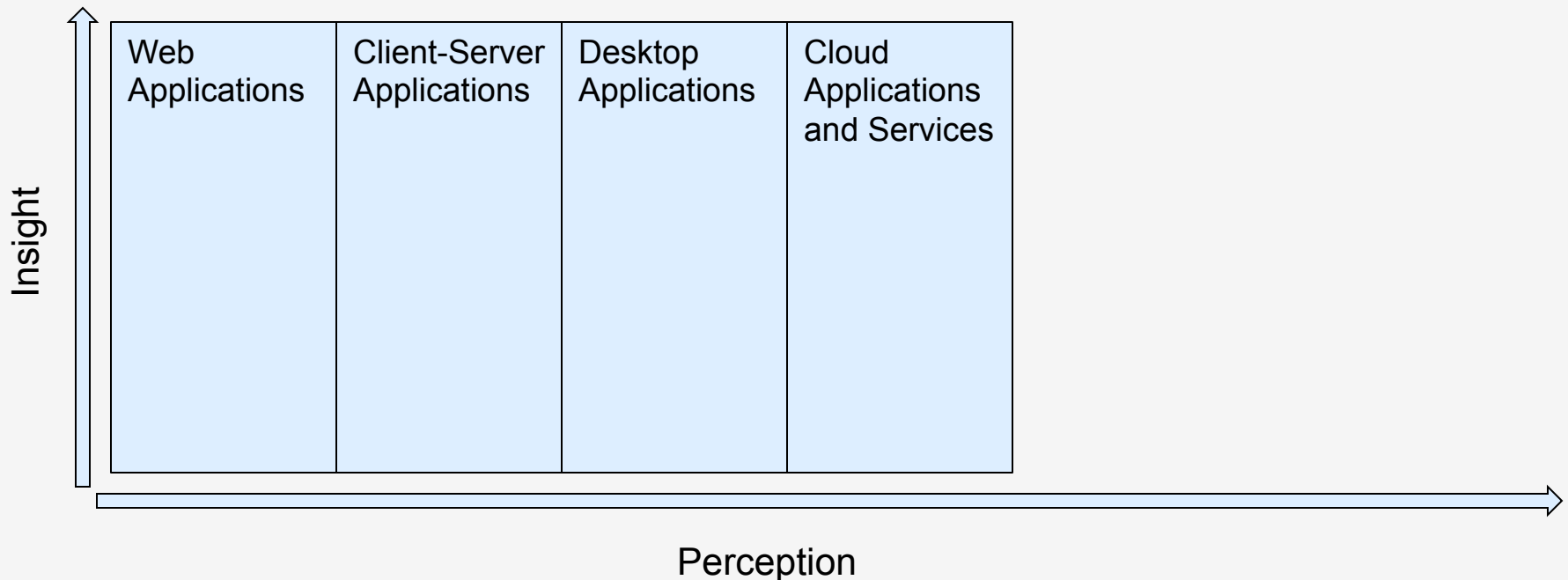
# Attack Surface: The Security Officer's Journey

- As perception of the problem of attack surface widens the scope of the problem increases



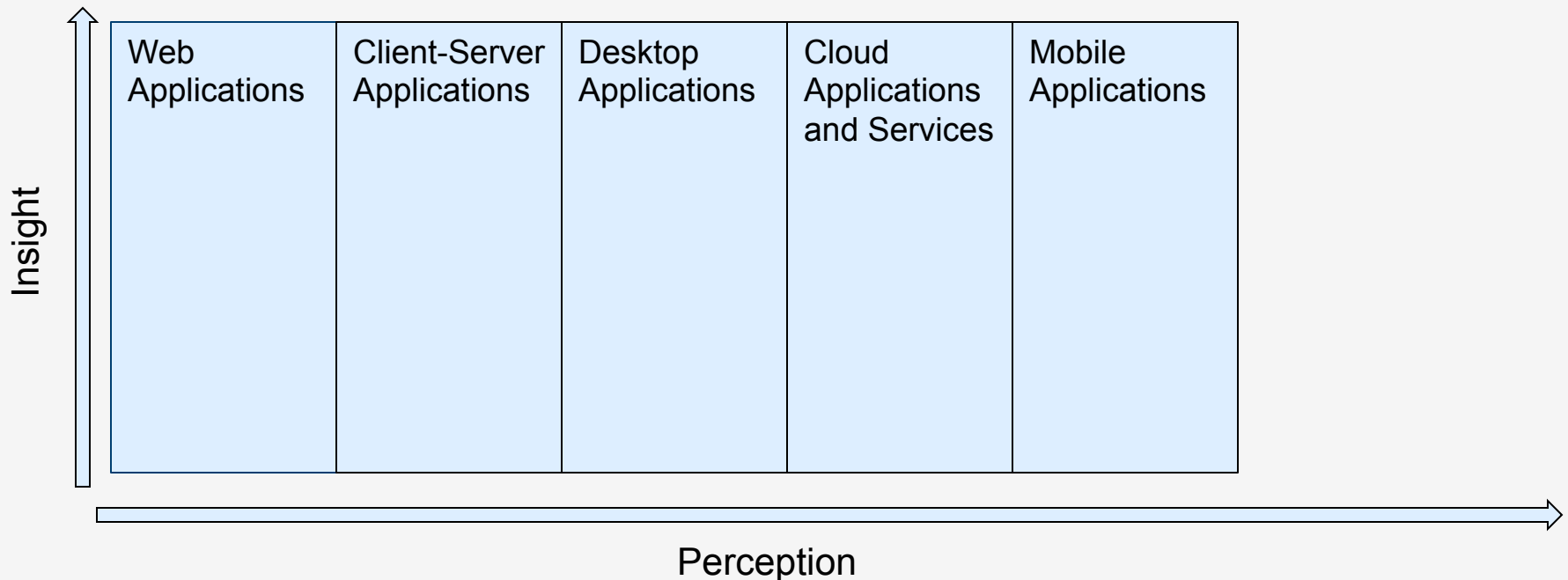
# Attack Surface: The Security Officer's Journey

- As perception of the problem of attack surface widens the scope of the problem increases



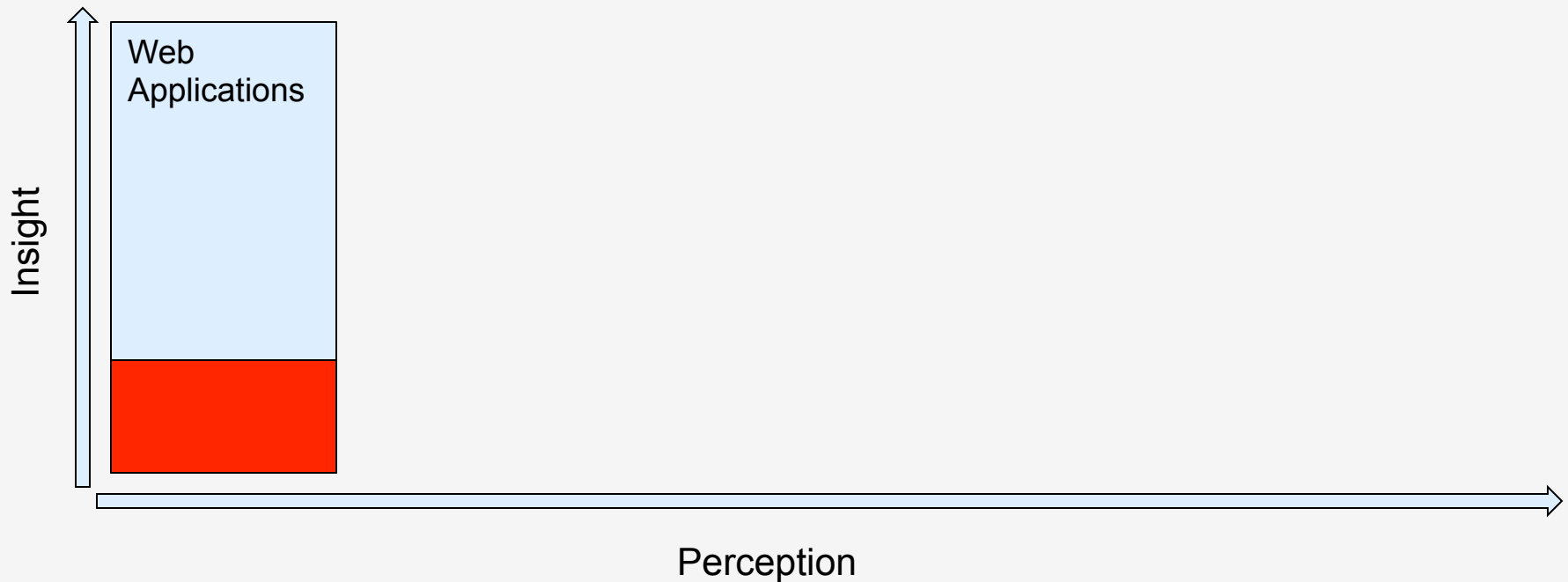
# Attack Surface: The Security Officer's Journey

- As perception of the problem of attack surface widens the scope of the problem increases



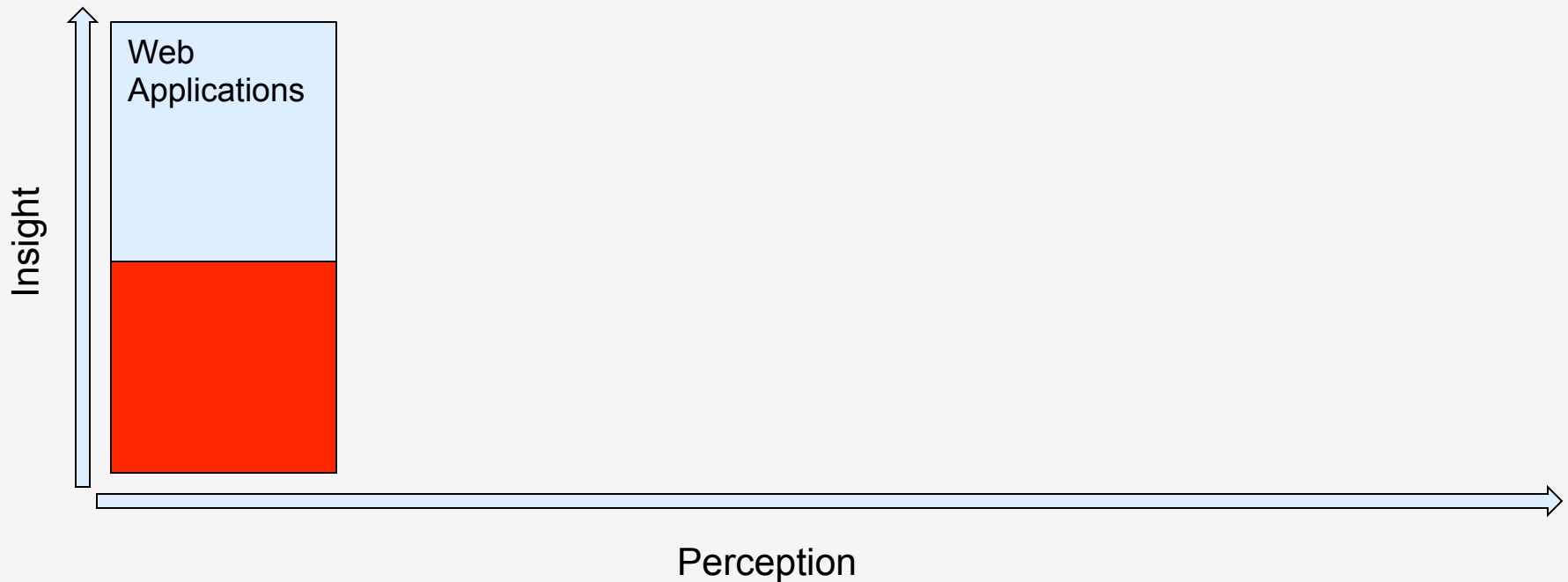
# Attack Surface: The Security Officer's Journey

- Discovery activities increase insight



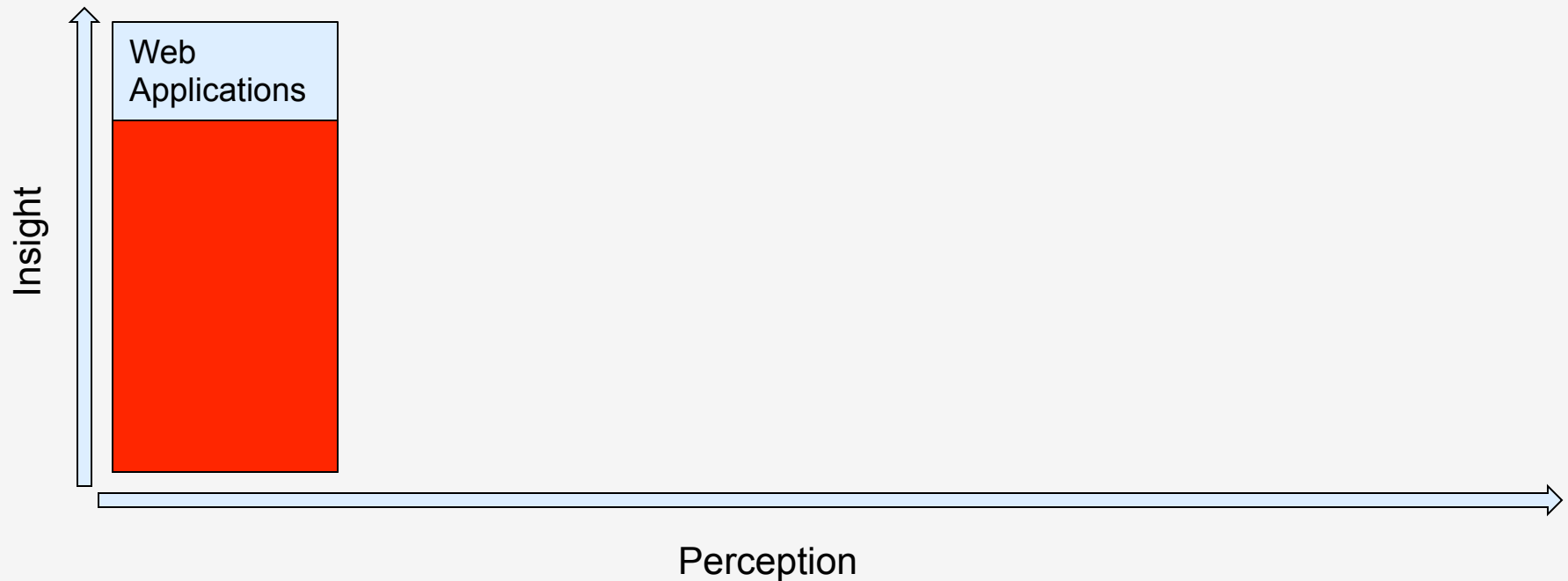
# Attack Surface: The Security Officer's Journey

- Discovery activities increase insight



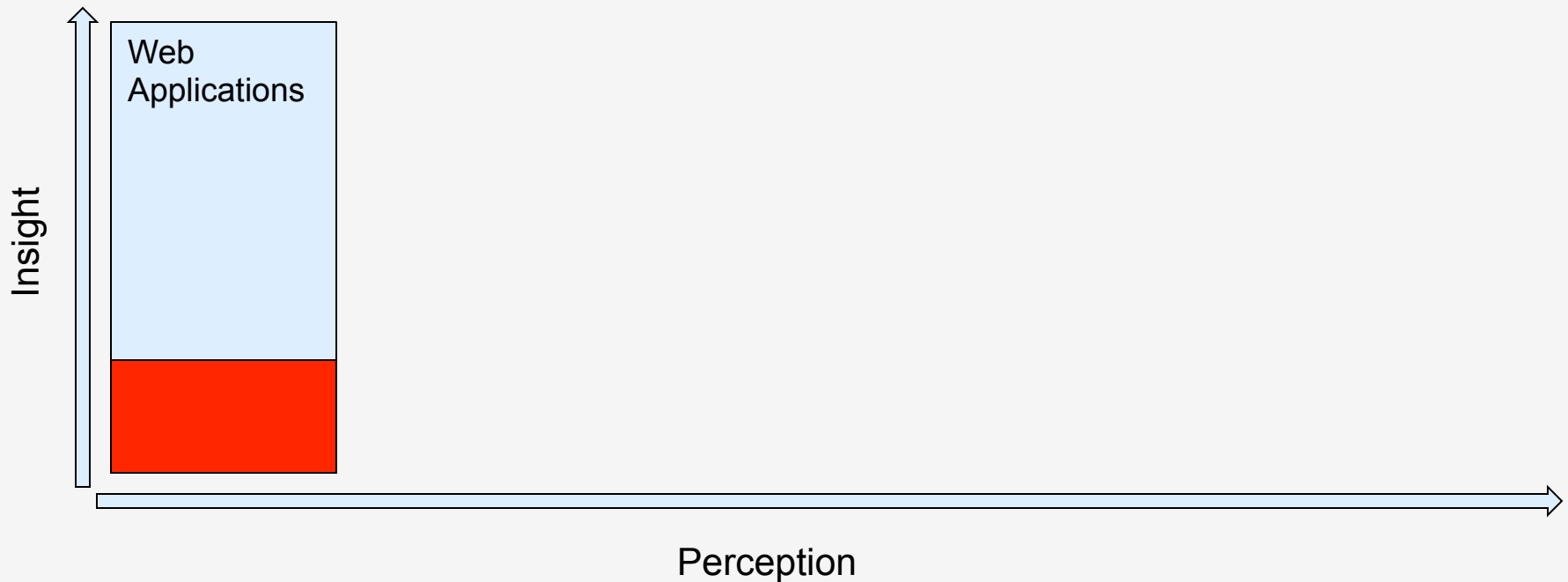
# Attack Surface: The Security Officer's Journey

- Discovery activities increase insight



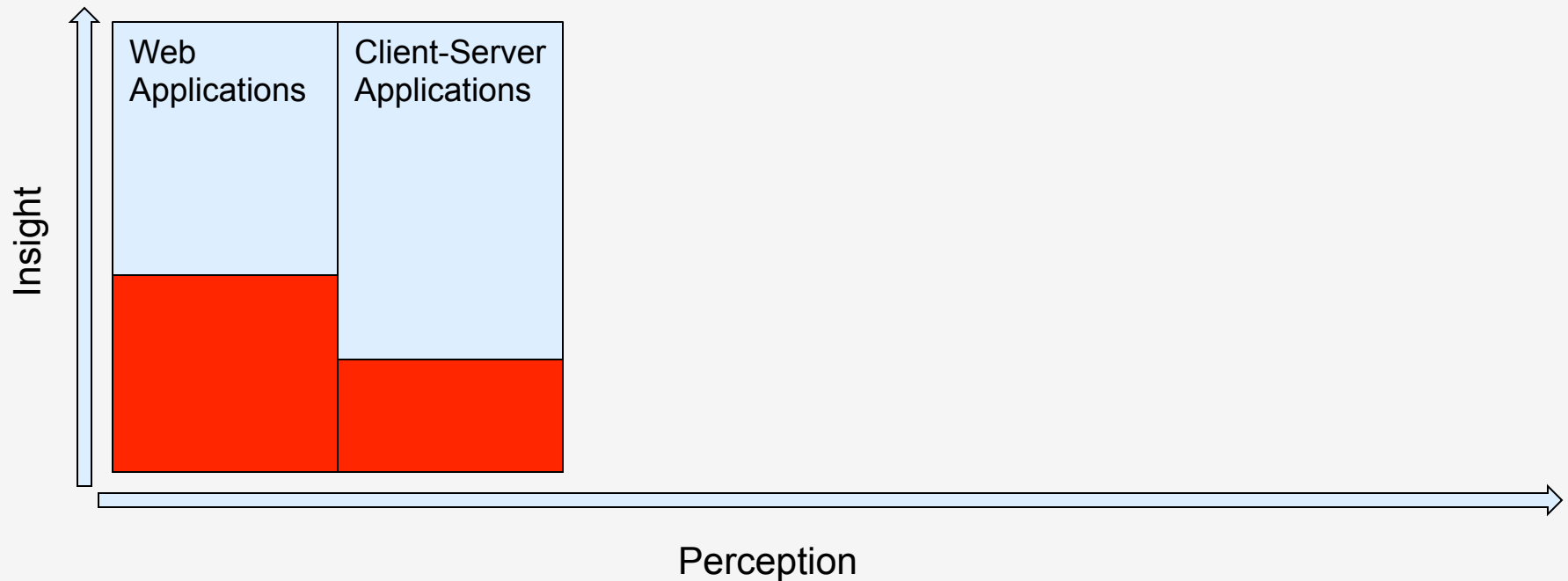
# Attack Surface: The Security Officer's Journey

- Over time you end up with a progression



# Attack Surface: The Security Officer's Journey

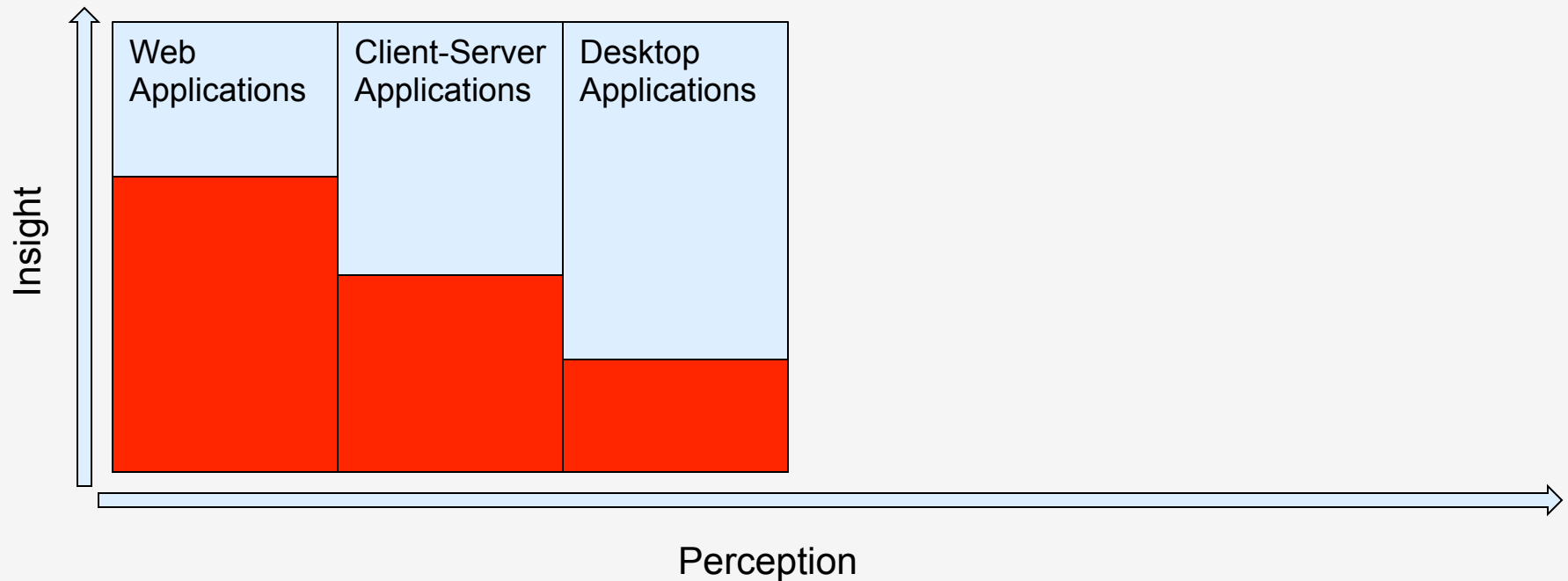
- Over time you end up with a progression





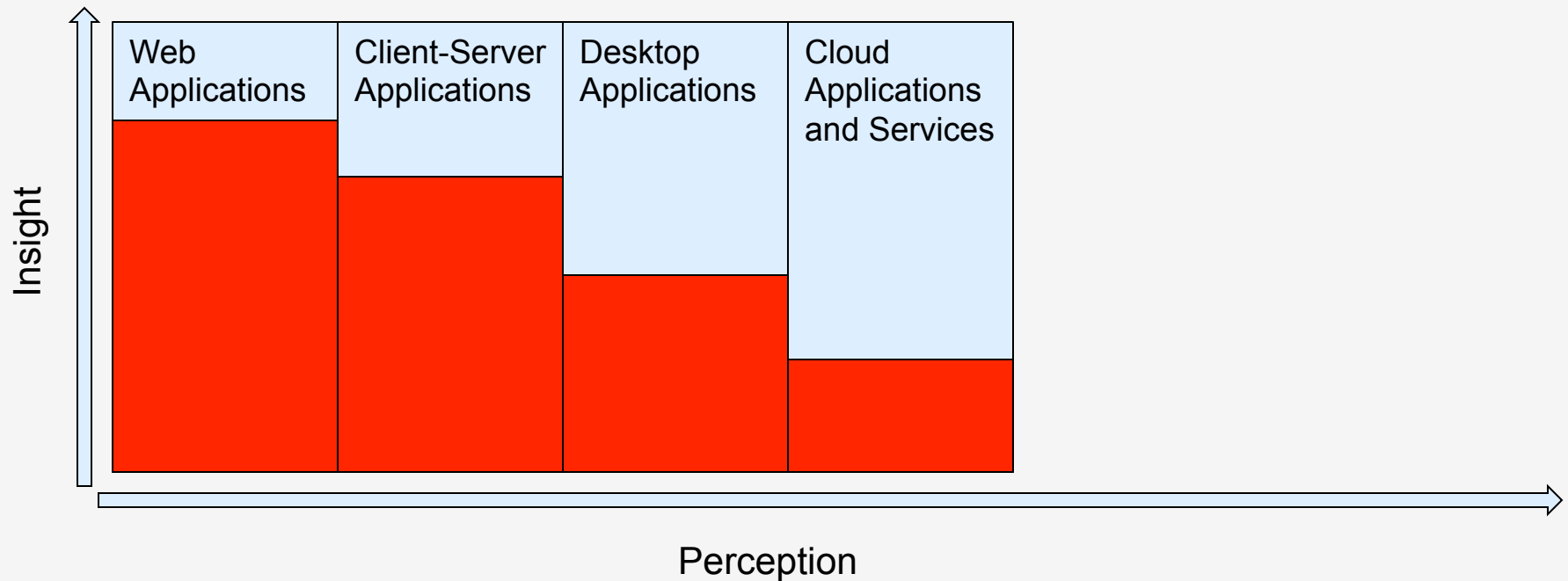
# Attack Surface: The Security Officer's Journey

- Over time you end up with a progression



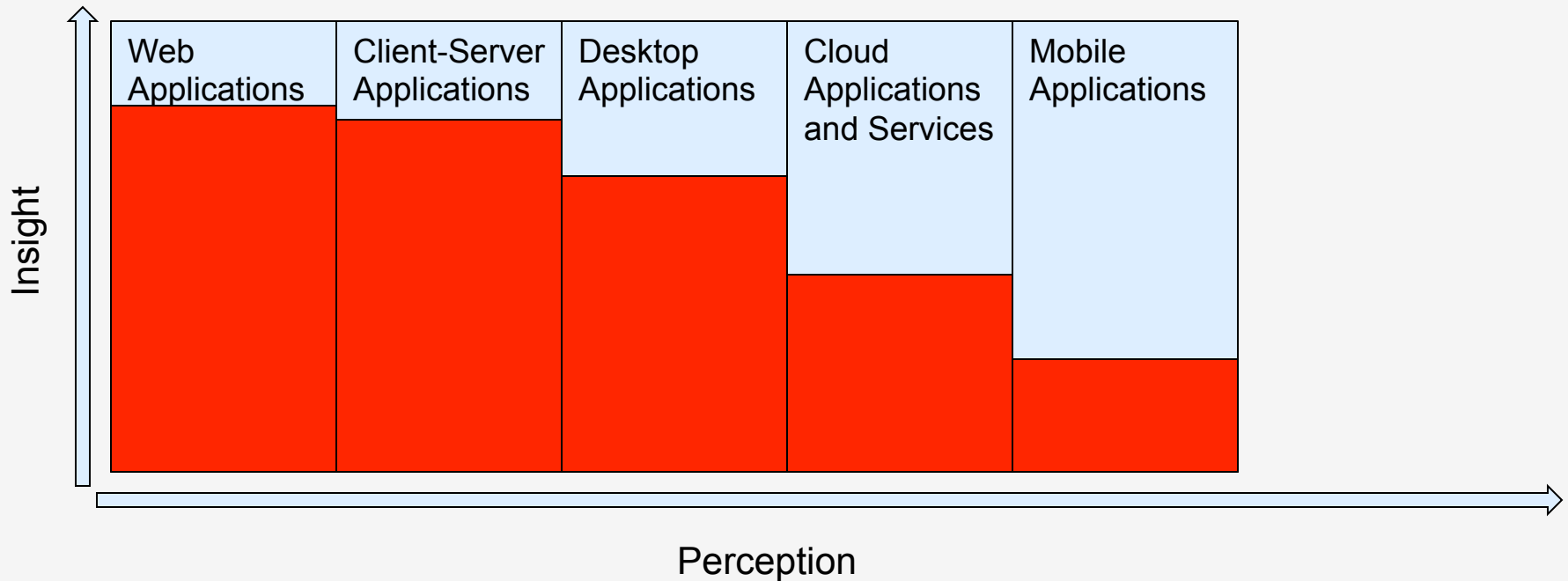
# Attack Surface: The Security Officer's Journey

- Over time you end up with a progression



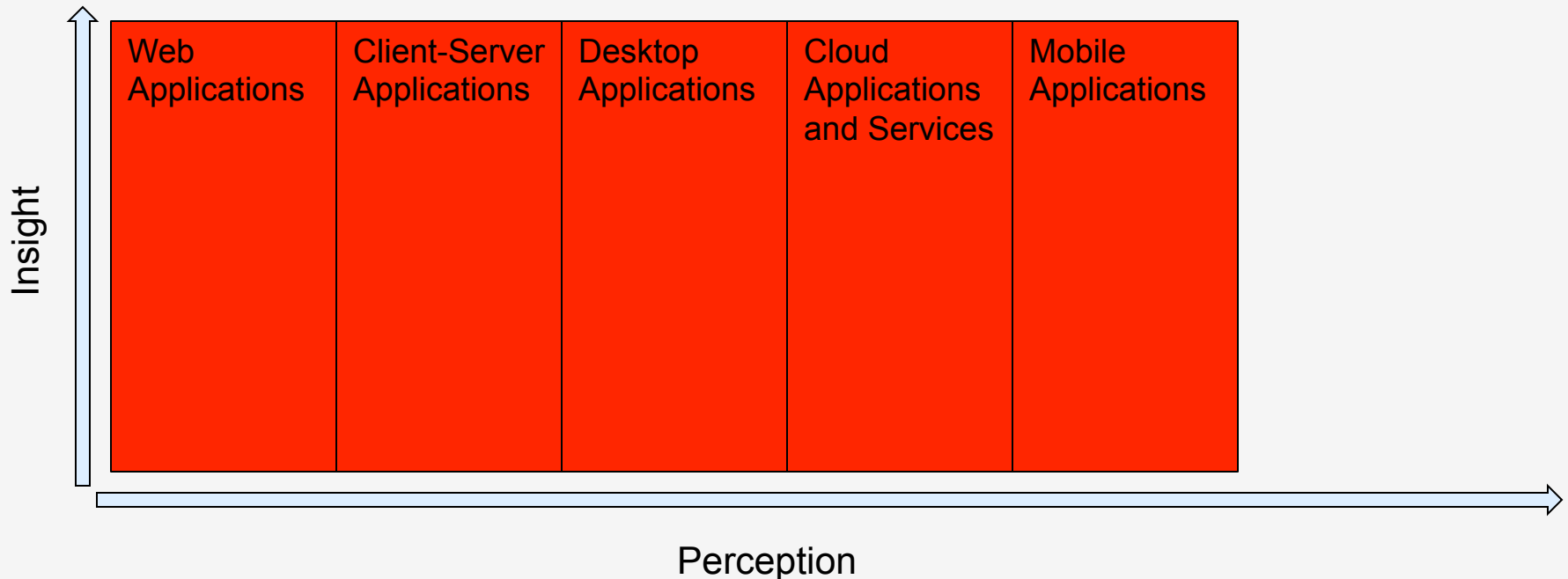
# Attack Surface: The Security Officer's Journey

- Over time you end up with a progression



# Attack Surface: The Security Officer's Journey

- When you reach this point it is called “enlightenment”
- You won't reach this point



# Value and Risk Are Not Equally Distributed

- Some Applications Matter More Than Others
  - Value and character of data being managed
  - Value of the transactions being processed
  - Cost of downtime and breaches
- Therefore All Applications Should Not Be Treated the Same
  - Allocate different levels of resources to assurance
  - Select different assurance activities
  - Also must often address compliance and regulatory requirements

# Do Not Treat All Applications the Same

- Allocate Different Levels of Resources to Assurance
- Select Different Assurance Activities
- Also Must Often Address Compliance and Regulatory Requirements

# What Goes Into An Application Test?

**An Application  
Test**

# What Goes Into An Application Test?

**Dynamic  
Analysis**

**Static  
Analysis**



# What Goes Into An Application Test?

**Automated  
Application  
Scanning**

**Static  
Analysis**

**Manual  
Application  
Testing**

# What Goes Into An Application Test?

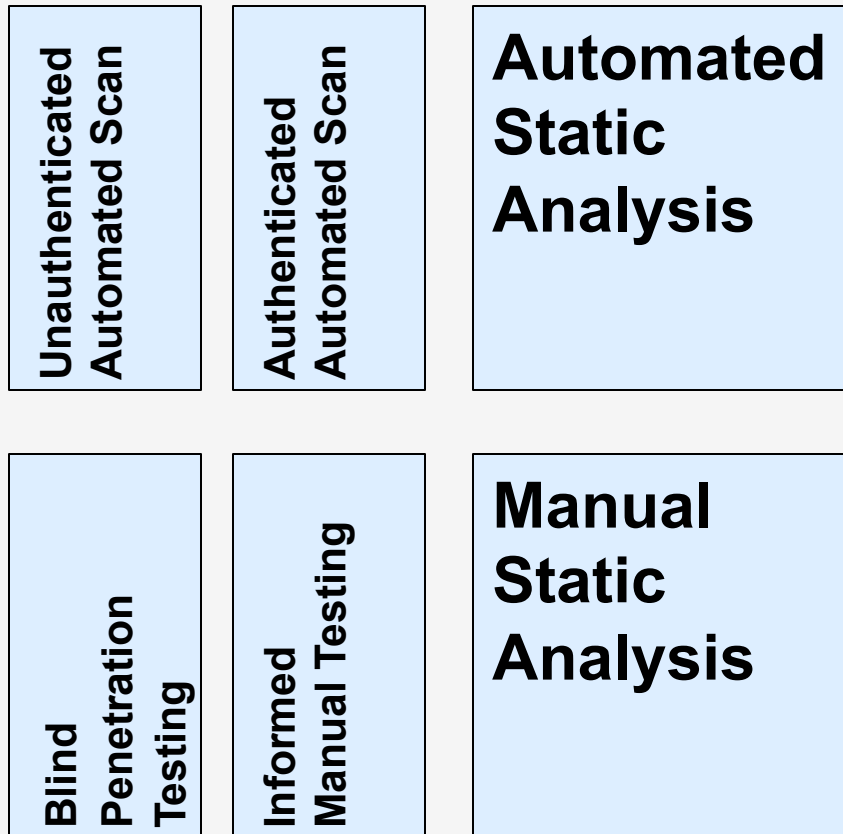
**Automated  
Application  
Scanning**

**Automated  
Static  
Analysis**

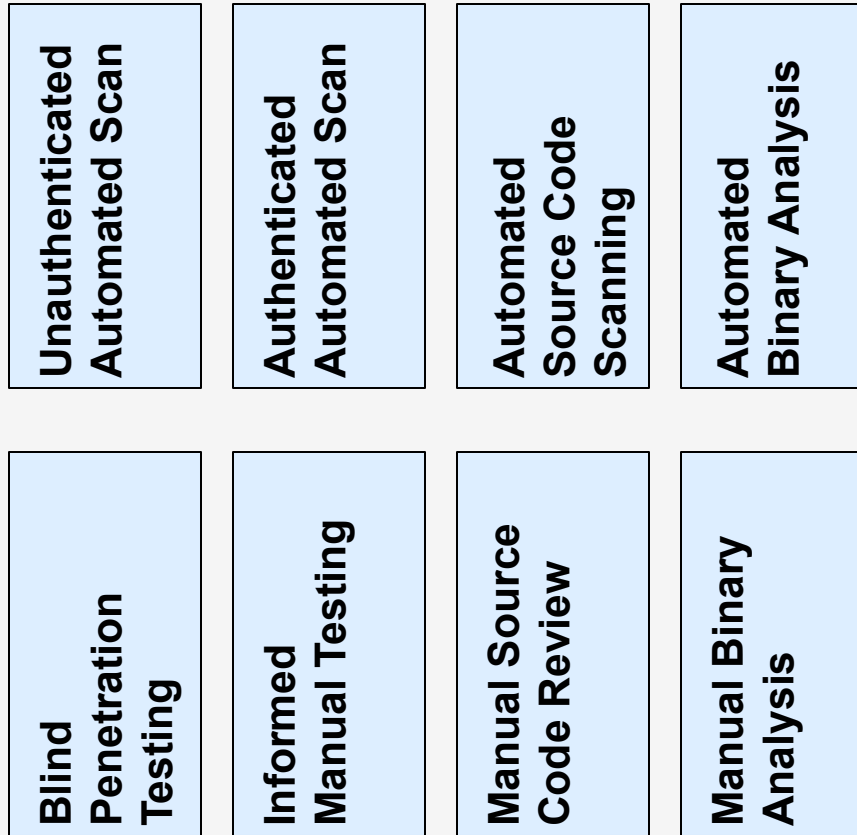
**Manual  
Application  
Testing**

**Manual  
Static  
Analysis**

# What Goes Into An Application Test?



# What Goes Into An Application Test?



# How To Allocate Scarce Resources?

- What Do You HAVE To Do?
  - What discretion do you have within these constraints?
- What Is Left Over?
- Strategies
  - Breadth-first
  - Depth-first
  - Hybrid

# Breadth-First

- Do Base-level Security Testing of Everything
  - Well, everything you can find
  - And everything you test with automation
- Automation is key
- Understand the limitations
  - Some applications cannot be effectively scanned
  - Often scans are unauthenticated
  - Whole classes of vulnerabilities are out of testing scope

# Depth-First

- Do Deeper Testing of Critical Applications
- Typically Combination of Automation and Manual Testing
- Understand the Limitations
  - Some applications remain unexamined
  - And breaches to those applications put shared resources and infrastructure at risk

# Hybrid

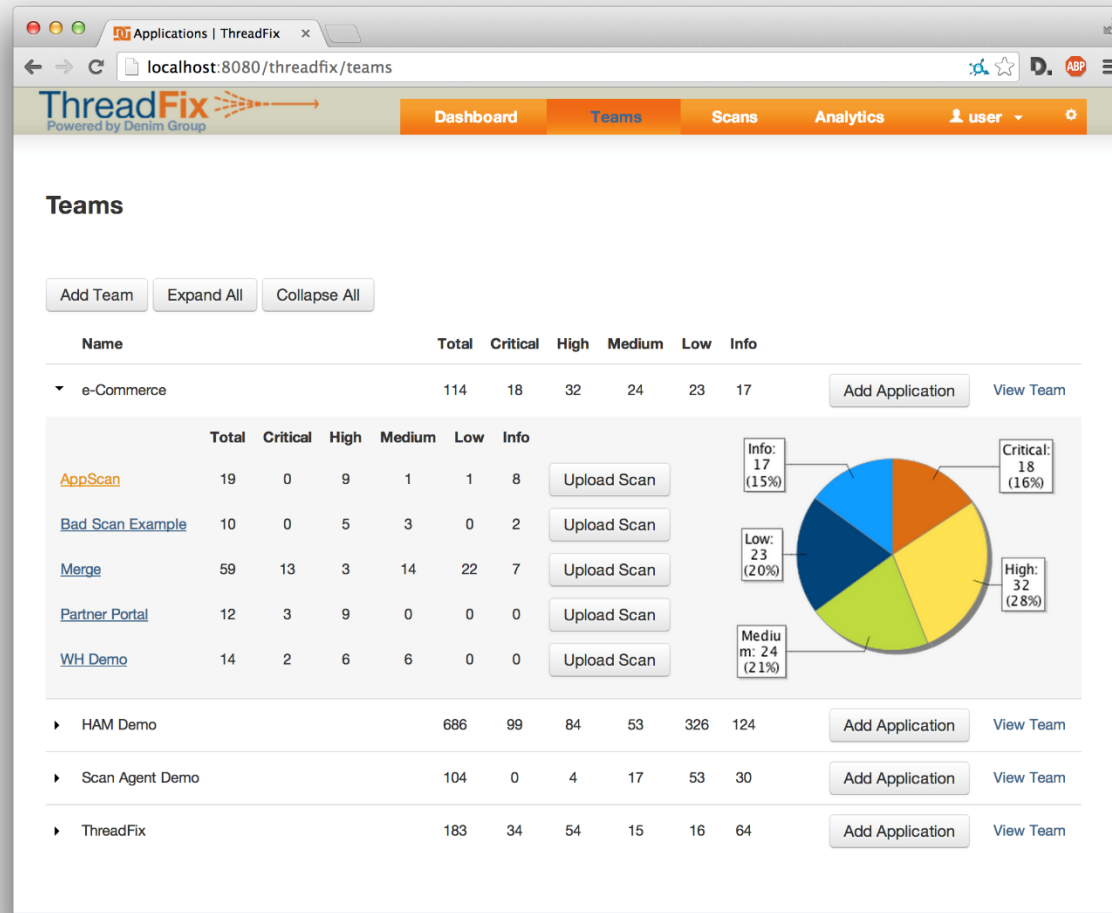
- Combination of Automation and Manual Testing Across Portfolio
- This is where most organizations end up
  - Often because regulatory and compliance mandates
- Know Your Gaps



# Application Portfolio Tracking

- Track multiple “Teams”
  - Arbitrary distinction – geography, line of business, common tools and practices
- Track multiple “Applications” per “Team”
  - Unit of scanning or testing
- Track Application metadata
  - Criticality, hosted URL, source code location
- Reporting can be done at the organization, Team or Application level

# Demo: Application Portfolio Tracking



# Fill ThreadFix Up With Vulnerability Data

- Manual file upload
- REST API
  - <https://github.com/denimgroup/threadfix/wiki/Threadfix-REST-Interface>
- Command Line Interface (CLI)
  - <https://github.com/denimgroup/threadfix/wiki/Command-Line-Interface>
  - JAR can also be used as a Java REST client library
- Jenkins plugin
  - Contributed from the ThreadFix community (yeah!)
  - <https://github.com/automationdomination/threadfix-plugin>

# What Does ThreadFix Do With Scan Results

- Diff against previous scans with same technology
  - What vulnerabilities are new?
  - What vulnerabilities went away?
  - What vulnerabilities resurfaced?
- Findings marked as false positive are remembered across scans
  - Hopefully saving analyst time
- Normalize and merge with other scanners' findings
  - SAST to SAST
  - DAST to DAST
  - SAST to DAST via Hybrid Analysis Mapping (HAM)

# Demo: Vulnerability Merge

The screenshot displays the ThreadFix web application interface. The browser address bar shows the URL: `localhost:8080/threadfix/organizations/2/applications/7/vulnerabilities/10670?nonce=B38E3E86CF70...`. The application header includes the ThreadFix logo (Powered by Denim Group) and navigation tabs: Dashboard, Teams, Scans, Analytics, and a user profile dropdown. The breadcrumb trail indicates the path: Applications Index / Team: e-Commerce / Application: Merge / Vulnerability 10670.

### Vulnerability Details

[Toggle More Info](#)

#### Basic Information

Type	Severity	Path	Parameter
Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') ( <a href="#">CWE Entry</a> )	Critical	/demo/Evallnjection2.php	command

#### Findings

Scanner Name	Severity	Vulnerability Type	Path	Calculated URL Path	Calculated Source Path	Parameter	Native ID
IBM Rational AppScan	High	Cross-Site Scripting	/demo/Evallnjection2.php	/demo/Evallnjection2.php	/demo/Evallnjection2.php	command	eb6aa5598d20a3468754f36b9d2755d6 <a href="#">View Finding</a>
Arachni	HIGH	Cross-Site Scripting (XSS)	/demo/Evallnjection2.php	/demo/Evallnjection2.php	/demo/Evallnjection2.php	command	1fb87c361a9a8e5e948461438440ab23 <a href="#">View Finding</a>
w3af	Medium	Cross site scripting vulnerability	/demo/Evallnjection2.php	/demo/Evallnjection2.php	/demo/Evallnjection2.php	command	70729a13c08ec2be98fedf7903f63ae2 <a href="#">View Finding</a>

#### Comments

User	Date	Comment
------	------	---------

# Hybrid Analysis Mapping (HAM)

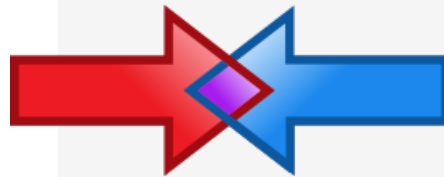
- Initial research funded by the US Department of Homeland Security (DHS) Science and Technology (S&T) Directorate via a Phase 1 and (now) Phase 2 Small Business Innovation Research (SBIR) contract
  - Acronyms!
- Initial goal: SAST to DAST merging
- Results: That, plus other stuff

# Hybrid Analysis Mapping – Phase 1 Goal

- Determine the feasibility of developing a system that can reliably and efficiently correlate and merge the results of automated static and dynamic security scans of web applications.



**HP Fortify SCA**



**IBM AppScan Standard**

# Dynamic Application Security Testing

- Spider to enumerate attack surface
- Fuzz to identify vulnerabilities based on analysis of request/response patterns



# Static Application Security Testing

- Use source or binary to create a model of the application
  - Kind of like a compiler or VM
- Perform analysis to identify vulnerabilities and weaknesses
  - Data flow, control flow, semantic, etc

```
String username = request.getParameter("username");
String sql = "SELECT * FROM User WHERE username = '" + username + "'";

Statement stmt;
stmt = con.createStatement();
stmt.execute(sql);
```

# Hybrid Analysis Mapping – Phase 1 Sub-Goals

- Standardize vulnerability types
- Match dynamic and static locations
- Improve static parameter parsing

# Hybrid Analysis Mapping

## Phase 1 - Technical Objectives

- **Technical Objective 1: Create common data structure standards for both automated static and dynamic security scanning results.**
  - Task 1: Create a Data Structure for Automated Dynamic Security Scanning Results
  - Task 2: Create a Data Structure for Automated Static Security Scanning Results
- **Technical Objective 2: Research and prototype methods of mapping the results of automated static and dynamic security scanning.**
  - Task 1: Create a Structured Model for Hybrid Analysis Mapping
  - Task 2: Investigate Approaches for Vulnerability Type Mapping
  - Task 3: Investigate Approaches for Mapping Source Code Files to URLs
  - Task 4: Investigate Approaches for Determining Injection Points

# Information Used

- Source Code (Git URL)
- Framework Type (JSP, Spring)
- Extra information from Fortify (if available)

# Vulnerability Types

- Successful CWE standardization
- Investigation into trees and Software Fault Patterns
  - Meant to correct for human errors
  - Hard to do in an automated fashion

# Unified Endpoint Database (Static and Dynamic)

- EndpointQuery
  - dynamicPath
  - staticPath
  - Parameter
  - httpMethod
  - codePoints [List<CodePoint>]
  - informationSourceType
- EndpointDatabase
  - findBestMatch(EndpointQuery query): Endpoint
  - findAllMatches(EndpointQuery query): Set<Endpoint>
  - getFrameworkType(): FrameworkType

# Parsing Attack Surface Locations

- JSP: Start with root JSP folder
- Spring: Parse @Controller classes

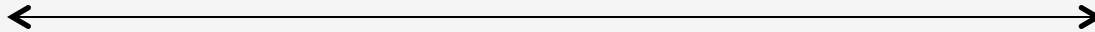
# Parsing Parameters

- JSP: Look for `request.getParameter()` calls
  - Coupled with lightweight dataflow analysis
- Spring: Parse `@RequestParam`, `@PathVariable`, `@Entity` annotations



# HAM Bridge

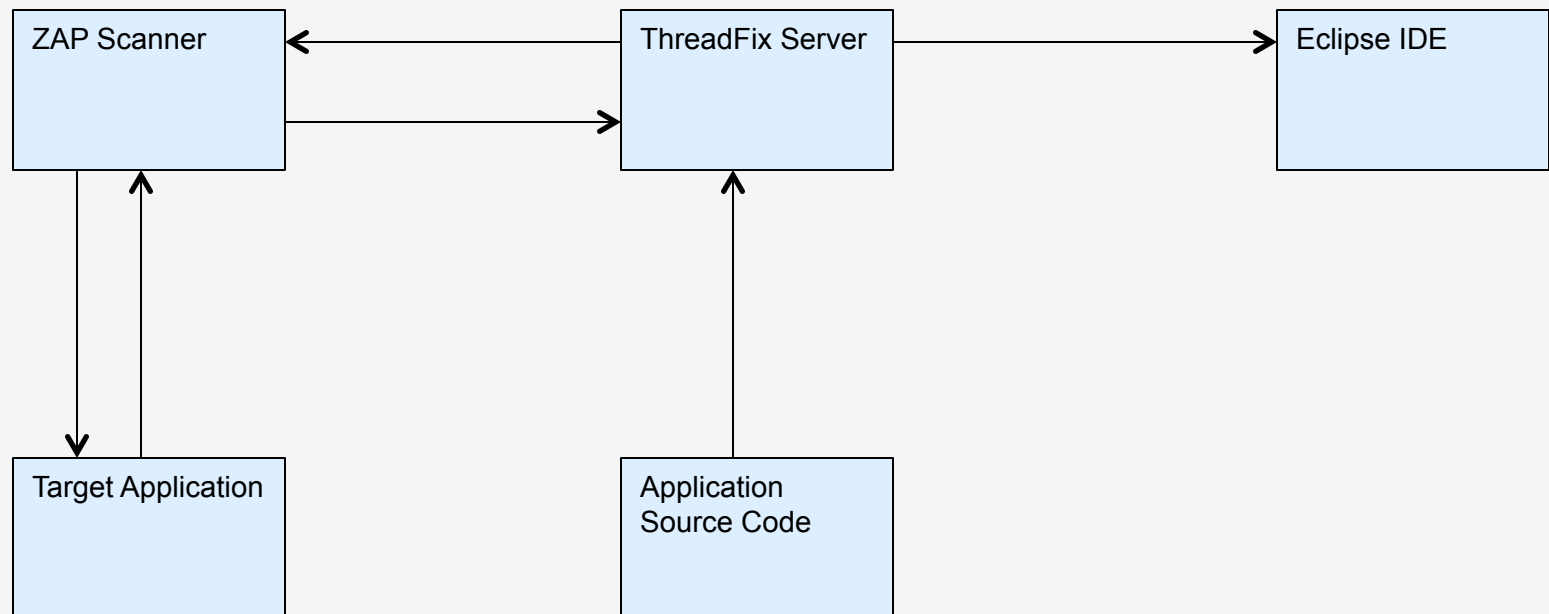
Static



Dynamic

- EndpointDatabase enables more than merging
- Scanner integration allows smarter scanning
- IDE plugin shows all vulnerabilities inline

# System Structure



# Demo: Merging Static and Dynamic Scanner Results

The screenshot displays a web application security scanner interface with two main sections: a list of vulnerabilities and a 'Scan History' table.

**Vulnerability List:**

Severity	Description	URL	Parameter	Action
Critical	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	/product.jsp		<a href="#">View More</a>
Critical	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	/register.jsp		<a href="#">View More</a>
Critical	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	/score.jsp		<a href="#">View More</a>
Critical	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	/search.jsp		<a href="#">View More</a>
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/basket.jsp	productid	<a href="#">View More</a>
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/contact.jsp	anticsrf	<a href="#">View More</a>
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/contact.jsp	comments	<a href="#">View More</a>

**Scan History:**

Channel	Scan Date	User
IBM Rational AppScan	6/19/13 9:18:47 AM	user
Fortify 360	6/19/13 12:41:39 AM	user

**Comments:**

User	Date	Comment
No comments found.		

[Add Comment](#)

**Additional Vulnerabilities (Bottom):**

Severity	Description	URL	Parameter	Action
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/contact.jsp	user1@thebodgeitstore.com	<a href="#">View More</a>
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/login.jsp	username	<a href="#">View More</a>
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/product.jsp	prodid	<a href="#">View More</a>
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/register.jsp	password1	<a href="#">View More</a>

# Demo: Merging Static and Dynamic Scanner Results

Vulnerability Details | ThreadFix | The Bodgeit Store

localhost:8080/threadfix/organizations/1/applications/1/vulnerabilities/28?nonce=7FEF0E60ECF6CABF857C29D3D5C6...

ThreadFix  
Powered by Denim Group

Dashboard Applications Scans Reports user

Applications Index / Team: HAM Demo / Application: Bodgeit / Vulnerability 28

### Vulnerability Details

Toggle More Info

#### Basic Information

Type	Severity	Path	Parameter
Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (CWE Entry)	Critical	/bodgeit/contact.jsp	comments

#### Findings

Scanner Name	Severity	Vulnerability Type	Path	Calculated URL Path	Calculated Source Path	Parameter	Native ID
IBM Rational AppScan	High	Stored Cross-Site Scripting	/bodgeit/contact.jsp	/contact.jsp	/contact.jsp	comments	aa113d21429a4e5fdb6a189c3ab23c79 <a href="#">View Finding</a>
Fortify 360	Critical	Cross-Site Scripting: Reflected	root/contact.jsp	/contact.jsp		comments	4AC2C441CFC0081776F519B90EBE6650 <a href="#">View Finding</a>

#### Data Flow

File Name root/contact.jsp

Line number 37

Line text String comments = (String) request.getParameter("comments");

File Name root/contact.jsp

Line number 37

# Merging Static and Dynamic Results Is Cool...

...But I want more

- Problem: Many DAST scanners handle applications with RESTful URLs poorly
- Problem: Many applications have “hidden” landing pages and parameters that will not be found by standard crawling
- Problem: DAST scanner results can be hard for developers to act on
- What else can we do with this attack surface model / database?
  - Clean up scanner results
  - Enumerate application attack surface
  - Map dynamic results to specific lines of code

# Demo: De-Duplicate Dynamic RESTful Scanner Results

The screenshot shows a web browser window with the address bar displaying `localhost:8080/threadfix/organizations/1/applications/4?nonce=CDB343B3C8A8F5428F447AAFDDA51CE8`. The page title is "PetClinic (vulnerable) | ThreadFix".

The main content area displays a list of vulnerabilities under the "Critical (11)" section. The table has columns: Severity, Type, Path, Parameter, and a "View More" link.

Severity	Type	Path	Parameter	View More
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/owners	lastName	<a href="#">View More</a>
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/owners/{id}/pets/{id}/visits/new	new	<a href="#">View More</a>
Critical	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	/owners/{id}/pets/{id}/edit	name	<a href="#">View More</a>
Critical	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	/owners/{id}/pets/new	birthDate	<a href="#">View More</a>
Critical	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	/owners/{id}/pets/{id}/visits/new	description	<a href="#">View More</a>

Below the vulnerability list, there are two sections: "Scan History" and "Comments".

**Scan History**

Channel	Scan Date	User
IBM Rational AppScan	6/4/13 10:57:15 AM	user
IBM Rational AppScan	6/4/13 10:57:15 AM	user

**Comments**

User	Date	Comment
1 user	10:44:04 11/20/2013	The app uses a REST-style URL scheme so AppScan is multi-reporting vulnerabilities. They seem to have been properly de-duped, however.

There is an "Add Comment" button below the comments section.

# Demo: De-Duplicate Dynamic RESTful Scanner Results

**Vulnerability Details** [Toggle More Info](#)

**Basic Information**

Type	Severity	Path	Parameter
Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') <a href="#">(CWE Entry)</a>	Critical	/petclinic/owners/10/pets/12/visits/new	description

**Findings**

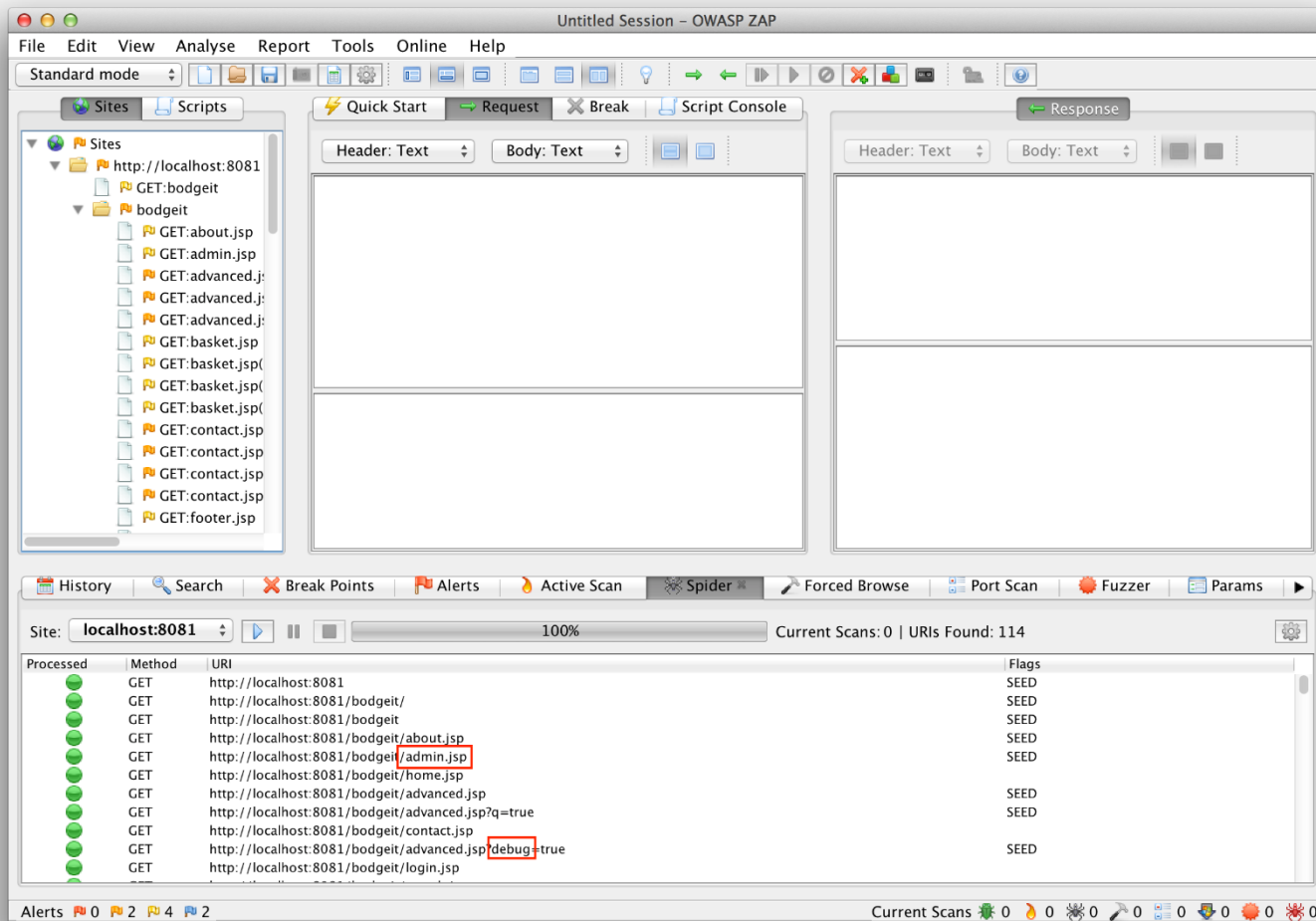
Scanner Name	Severity	Vulnerability Type	Path	Calculated URL Path	Calculated Source Path	Parameter	Native ID
IBM Rational AppScan	High	Blind SQL Injection	/petclinic/owners/10/pets/12/visits/new	/owners/{id}/pets/{id}/visits/new	/src/main/java/org/springframework/sample/petclinic/web/VisitController.java	description	1a07468c95c11f782dee9ab5b7fb6c05 <a href="#">View Finding</a>
IBM Rational AppScan	High	Blind SQL Injection	/petclinic/owners/4/pets/5/visits/new	/owners/{id}/pets/{id}/visits/new	/src/main/java/org/springframework/sample/petclinic/web/VisitController.java	description	4054c867b8edad825d4e4468b3e7ac9a <a href="#">View Finding</a>

# Demo: Application Attack Surface (CLI)

```
ham — bash — 124x40
Dans-MacBook-Pro:ham dcornell$ java -jar endpoints-experimental-20131106.jar ~/git/bodgeit/root/
INFO [main] MergeConfigurationGenerator.getDatabase(20) | Attempting to calculate framework type based on project contents.
INFO [main] MergeConfigurationGenerator.getType(16) | Attempting to guess Framework Type from source tree.
INFO [main] MergeConfigurationGenerator.getType(17) | File: /Users/dcornell/git/bodgeit/root
INFO [main] ServletMappings.guessApplicationType(175) | About to guess application type from web.xml.
INFO [main] ServletMappings.guessApplicationType(217) | Determined that the framework type was JSP
INFO [main] MergeConfigurationGenerator.getType(34) | Source tree framework type detection returned: JSP
INFO [main] MergeConfigurationGenerator.getDatabase(24) | Calculated framework : JSP.
INFO [main] GeneratorBasedEndpointDatabase.<init>(54) | Using generic EndpointGenerator-based translator.
INFO [main] GeneratorBasedEndpointDatabase.buildMappings(69) | Building mappings.
INFO [main] GeneratorBasedEndpointDatabase.buildMappings(82) | Done building mappings. Static keys: 0, dynamic keys: 16
[POST, GET] /about.jsp,[]
[POST, GET] /admin.jsp,[]
[POST, GET] /advanced.jsp,[q, debug]
[POST, GET] /basket.jsp,[update, productid, quantity, debug]
[POST, GET] /contact.jsp,[anticsrf, debug, comments]
[POST, GET] /footer.jsp,[]
[POST, GET] /header.jsp,debug
[POST, GET] /home.jsp,debug
[POST, GET] /init.jsp,[]
[POST, GET] /login.jsp,[username, debug, password]
[POST, GET] /logout.jsp,[]
[POST, GET] /password.jsp,[password1, password2]
[POST, GET] /product.jsp,[typeid, prodid, debug]
[POST, GET] /register.jsp,[password1, username, password2, debug]
[POST, GET] /score.jsp,debug
[POST, GET] /search.jsp,[q, debug]
Dans-MacBook-Pro:ham dcornell$
```



# Demo: Seed Scanner with Attack Surface



# **Prioritize application risk decisions based on data**

# Vulnerability Filtering

- Filter vulnerability data
  - Scanner, scanner count
  - Vulnerability type
  - Path, parameter
  - Severity
  - Status
  - Aging
- Save filters for future use

# Demo: Vulnerability Filtering

The screenshot displays the ThreadFix Analytics web application. The browser address bar shows `localhost:8080/threadfix/reports`. The application header includes the ThreadFix logo (Powered by Denim Group) and navigation tabs: Dashboard, Teams, Scans, Analytics (selected), and a user profile dropdown. The main content area is titled "Analytics" and features tabs for Trending, Snapshot, Comparison, and Vulnerability Search (selected). Under the Vulnerability Search tab, the "Results" section shows a list of vulnerabilities. The first two are "Critical" with a count of 52, and the third is "Cross-site Scripting" with a count of 52. A pagination bar shows 10, 25, 50, 100, and a "Last" button. The results list includes details for each vulnerability: Application, Path, Parameter, and the scanning tool (IBM Rational AppScan). The "Filters" section on the right has tabs for "Filters" and "Load Filters". A dropdown menu shows "XSS over 90". Below this are buttons for "Delete Selected Filter" and "Clear Filter".

**ThreadFix**  
Powered by Denim Group

Dashboard Teams Scans **Analytics** user

**Analytics**

Trending Snapshot Comparison **Vulnerability Search**

**Results**

- Critical 52
- 52 Cross-site Scripting

10 25 50 100 First 1 2 3 4 5 Last

**Application** / Merge Example - Bodgelt  
**Path** /basket.jsp  
**Parameter** productid  
IBM Rational AppScan  
0 0 View More

**Application** / Bodgelt  
**Path** /basket.jsp  
**Parameter** productid  
IBM Rational AppScan  
0 0 View More

**Application** / Merge Example - Bodgelt  
**Path** /contact.jsp  
**Parameter** user1@thebodgeitstore.com  
IBM Rational AppScan  
0 0 View More

**Filters**

Filters Load Filters

XSS over 90

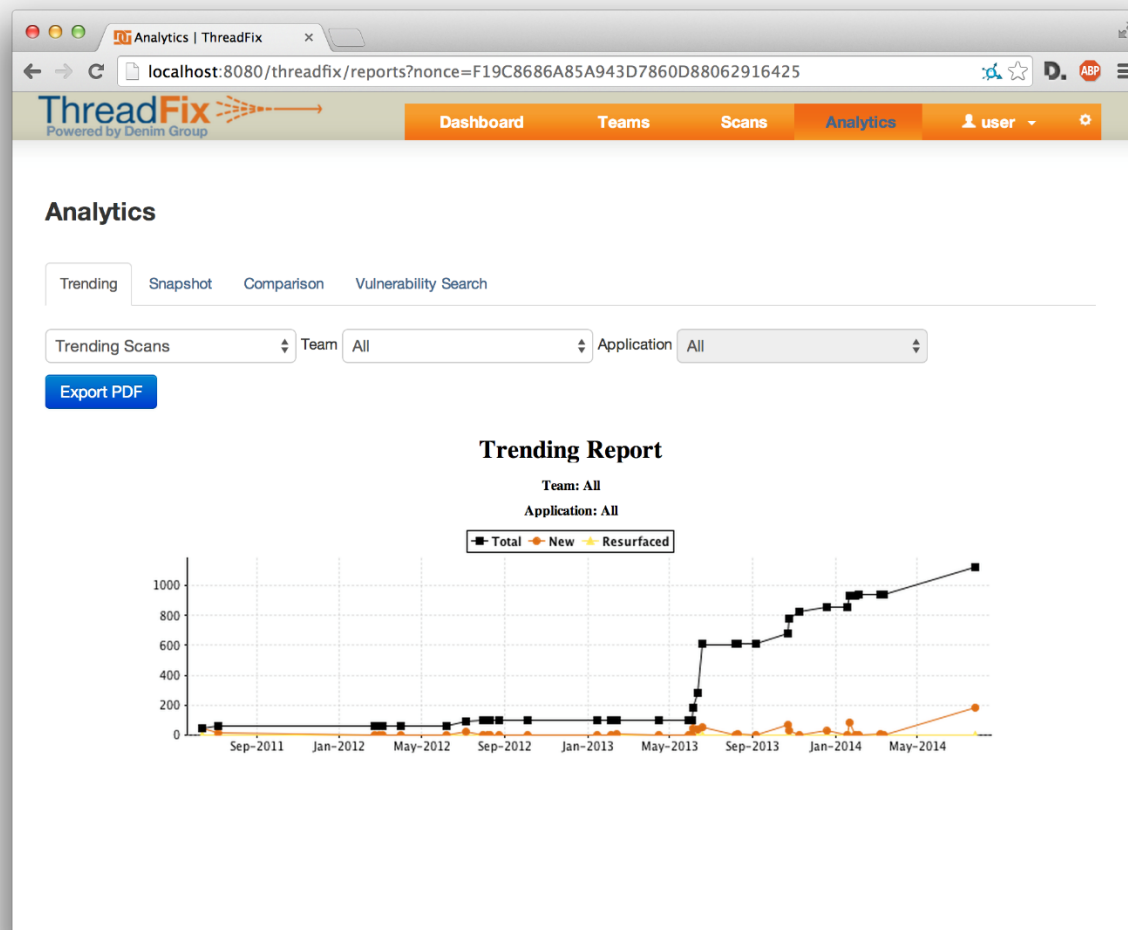
Delete Selected Filter

Clear Filter

# Reporting

- Trending
- Progress by Vulnerability
  - For program benchmarking
- Portfolio Report
  - For resource prioritization
- Comparison
  - For scanner/technology benchmarking

# Demo: Reporting



**Translate vulnerabilities  
to developers in the  
tools they are already  
using**

# Mapping Vulnerabilities to Defects

- 1:1 mapping is (usually) a horrible idea
  - 500 XSS turned into 500 defects?
  - If it takes longer to administer the bug than it does to fix the code...
- Cluster like vulnerabilities
  - Using the same libraries / functions
  - Cut-and-paste remediation code
  - Be careful about context-specific encoding
- Combine by severity
  - Especially if they are cause for an out-of-cycle release
- Which developer “owns” the code?



# Defect Tracker Integration

- Bundle multiple vulnerabilities into a defect
  - Using standard filtering criteria
- ThreadFix periodically updates defect status from the tracker

# Demo: Defect Tracker Integration

The screenshot displays the ThreadFix Partner Portal interface. The browser address bar shows the URL: `localhost:8080/threadfix/organizations/2/applications/2?nonce=EBA570CE004BAADD1330D55D962F1...`. The page is divided into a 'Results' section on the left and a 'Filters' section on the right.

**Results Section:**

- Critical (3):**
  - SQL Injection (2):**
    - Two identical entries are shown. Each entry includes a 'Path' of `/owners`, a 'Parameter' of `lastName`, and a 'File' of `java/org/springframework/samples/petclinic/repository/jpa/JpaOwnerRepositoryImpl.java`. Each entry is associated with 'Fortify 360' and 'Issue THREAD-145 (Open)'. Below each entry are icons for a comment, a document, and a 'View More' link.
- High (9):**
  - Authentication Bypass by Spoofing (1):**
  - Authorization Bypass Through User-Controlled SQL Primary Key (6):**
    - One entry is shown with a 'Path' of `/owners/{id}`, a 'Parameter' of `ownerId`, and a 'File' of `java/org/springframework/samples/petclinic/repository/jpa/JpaOwnerRepositoryImpl.java`. It is associated with 'Fortify 360' and 'Issue THREAD-146 (Closed)'. Below the entry are icons for a comment, a document, and a 'View More' link.

**Filters Section:**

- Buttons: 'Filters', 'Load Filters', 'Expand All', 'Clear'.
- Expandable filters (each with a '+' icon):
  - Scanner and # Merged
  - Field Controls
  - Aging
  - Date Range
  - Save Current Filter
- Button: 'Export CSV'.

# IDE Plug Ins

- Import vulnerability data to integrated development environments (IDEs)
- Static (SAST) scanners
  - Easy
- Dynamic (DAST) scanners
  - Possible using Hybrid Analysis Mapping (HAM)

# Map Dynamic Scan Results to LoC in IDE

The screenshot shows the Spring Tool Suite IDE with the file `JpaVisitRepositoryImpl.java` open. The code contains a `findByPetId` method that constructs a SQL query string. The variable `petId` is highlighted with a red box. Below the code editor, the ThreadFix Vulnerabilities panel is visible, showing a list of detected vulnerabilities.

```
private EntityManager em;

@Override
public void save(Visit visit) {
    if (visit.getId() == null) {
        this.em.persist(visit);
    }
    else {
        this.em.merge(visit);
    }
}

@Override
@SuppressWarnings("unchecked")
public List<Visit> findByPetId(Integer petId) {
    Query query = this.em.createQuery("SELECT visit FROM Visit v where v.pets.id= " + petId);
    return query.getResultList();
}
```

0 errors, 219 warnings, 0 others (Filter matched 100 of 219 items)

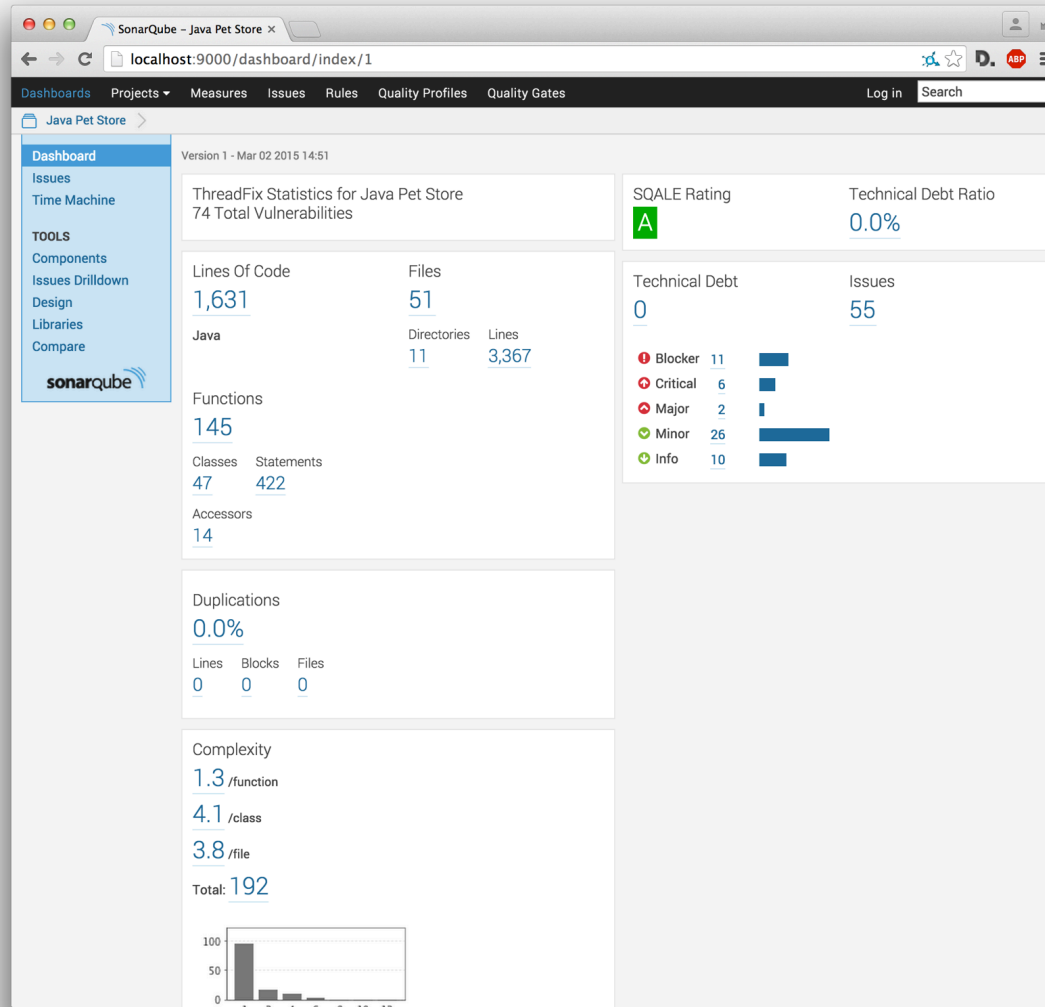
Resource	Location	Parameter	CWE	CWE Name	Defect Url
JpaOwnerR...	line 64	null	566	Authorization Bypass Through User-Controlled SQL Primary Key	
JpaOwnerR...	line 64	null	89	Improper Neutralization of Special Elements used in an SQL...	
JpaOwnerR...	line 64	ownerId	566	Authorization Bypass Through User-Controlled SQL Primary Key	
JpaOwnerR...	line 64	ownerId	566	Authorization Bypass Through User-Controlled SQL Primary Key	
JpaOwnerR...	line 64	ownerId	566	Authorization Bypass Through User-Controlled SQL Primary Key	
JpaVisitRep...	line 60	null	566	Authorization Bypass Through User-Controlled SQL Primary Key	
JpaVisitRep...	line 60	null	89	Improper Neutralization of Special Elements used in an SQL...	
JpaVisitRep...	line 60	pet	566	Authorization Bypass Through User-Controlled SQL Primary Key	
OwnerCont	line 84	dri	550	Information Exposure Through Server Error Message	

Writable Smart Insert 60 : 98

# SonarQube Integration

- Pull security vulnerabilities into the backlog that is getting tracked on SonarQube
- Can be done either via the ThreadFix server or by analyzing local files
- This is essentially a universal security tool plugin for SonarQube

# SonarQube Integration



# Important Links

- Main ThreadFix website: [www.threadfix.org](http://www.threadfix.org)
  - General information, downloads
- ThreadFix GitHub site: [www.github.com/denimgroup/threadfix](http://www.github.com/denimgroup/threadfix)
  - Code, issue tracking
- ThreadFix GitHub wiki: <https://github.com/denimgroup/threadfix/wiki>
  - Project documentation
- ThreadFix Google Group:  
<https://groups.google.com/forum/?fromgroups#!forum/threadfix>
  - Community support, general discussion

## Questions / Contact Information

### Dan Cornell

Principal and CTO

[dan@denimgroup.com](mailto:dan@denimgroup.com)

Twitter @danielcornell

(844) 572-4400

[www.denimgroup.com](http://www.denimgroup.com)

[www.threadfix.org](http://www.threadfix.org)