

Code Pulse

SwA Visual Analytics for Dynamic Analysis of Code

BAA-11-02-TTA 01-0066-I



DHS Cyber Security Division
2012 Principal Investigator's Meeting

October 11th 2012

Hassan Radwan
Principal Investigator

Hassan.Radwan@securedisions.com
518 207 3106



Code Pulse Overview

Goal

Improve static source code analysis workflows by introducing execution context to the analysis process.

Objective

The outcome will be a software application used in conjunction with static analysis tools to visually depict execution context of vulnerabilities to prioritize critical code paths. The software will fuse dynamic traces and static analysis into one seamless system automating code context processing and improving communication between stakeholders.

Scope

- TTA #1: Software Assurance
- 20 Month Period of Performance, Ending April 2014
- Targeting Java-based codebases

About Secure Decisions

Division of Applied Visions, Inc.

Founded 1987

40 people in Northport and Clifton Park, NY

Commercial software product development

- Hallmark, Reuters, AC Nielsen

Significant visualization experience

- Software Vis, Cyber Security Vis, Military C2 Vis

Secure Decisions security division launched in 2000

- Cyber decision analysis, Visual analytics of network data, Visual systems for cyber training, Mission impact of cyber attacks



Dynamic Analysis vs Dynamic Tracing

Project title: **SwA Visual Analytics for Dynamic Analysis of Code**

Project working title: **Code Pulse**

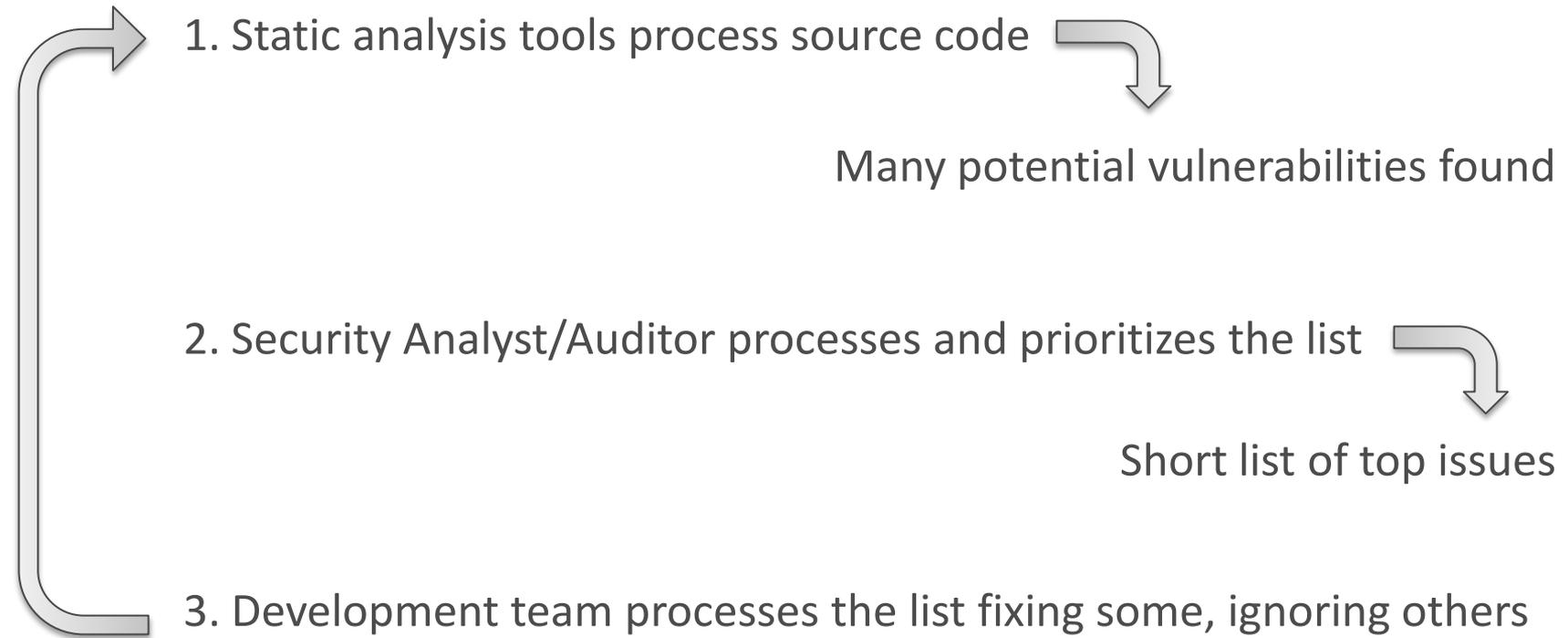
Dynamic Analysis

- Broad term
- In SwA used to describe the process to detect potential vulnerabilities at runtime (A.K.A. Penetration Testing, Black Box Testing)
- Source code context not available

Dynamic Tracing

- Monitor runtime execution
- Used by profiling tools
- Identify which methods are called, call graph, call durations, and call frequencies

Typical Static Analysis Workflow



Automation Mismatch

Data overload: Static analysis tools identify thousands of potential vulnerabilities

STATIC ANALYSIS



Automated Process



VULNERABILITY TRIAGE

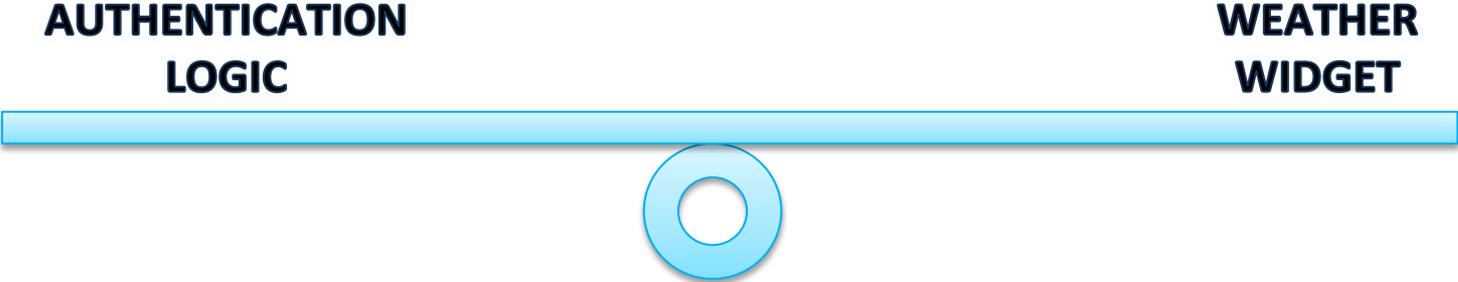


Manual Process

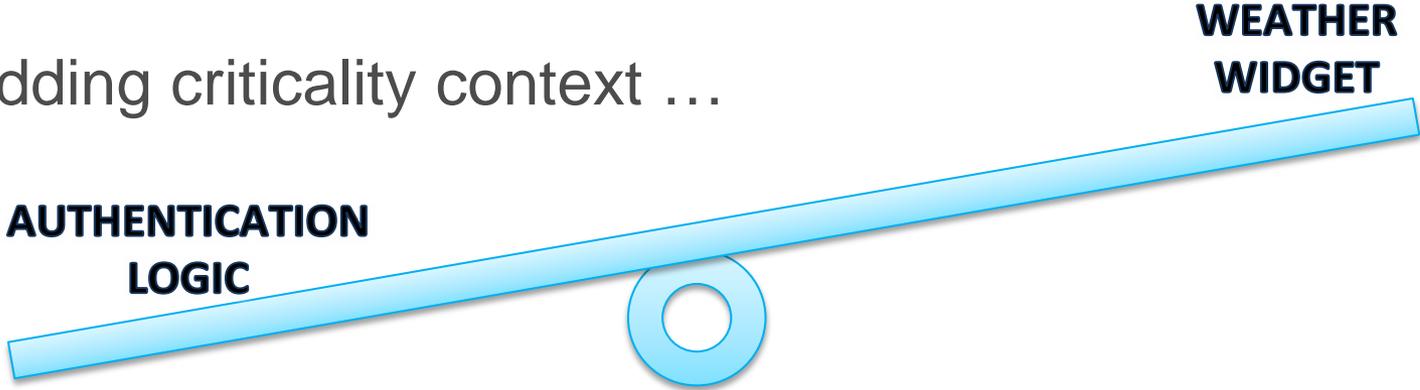
Identification process is automated, triage is still manual

Code (In)Equality

Static analysis tools are equal opportunity assessors



Adding criticality context ...



Code is not equal; vulnerabilities in critical code paths need higher prioritization

Vulnerability-First or Code-First?

Static analysis tools present results using a vulnerability-first approach


SECURITY ANALYST



Filter results by:

▼ Defect Type:

- Any
- High Impact:
- Medium Impact: only
- Low Impact:

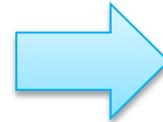
▶ API usage errors

▶ Class hierarchy inconsistencies

▶ Null pointer dereferences

▶ Resource leaks

VULNERABILITY GROUPINGS




INDIVIDUAL VULNERABILITY

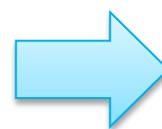
Developers use a code-first approach to think of their codebases


DEVELOPER



- ▶  junit.extensions
- ▶  junit.framework
 - ▶  Assert.java
 - ▶  AssertionError.java
 - ▶  AssertionError
 - ▶ serialVersionUID
 - ▶ AssertionError()
 - ▶ AssertionError(Str)
 - ▶  ComparisonCompactor.java

SOURCE CODE GROUPINGS




INDIVIDUAL VULNERABILITY

Security specialists and development teams have different mental models yet both need to communicate effectively for the shared SwA objective

Approach



Approach

1

Decrease manual tedium in weakness triage process

AUTOMATION

Provide the tools to quickly and effectively identify critical code paths

PRIORITIZATION

2

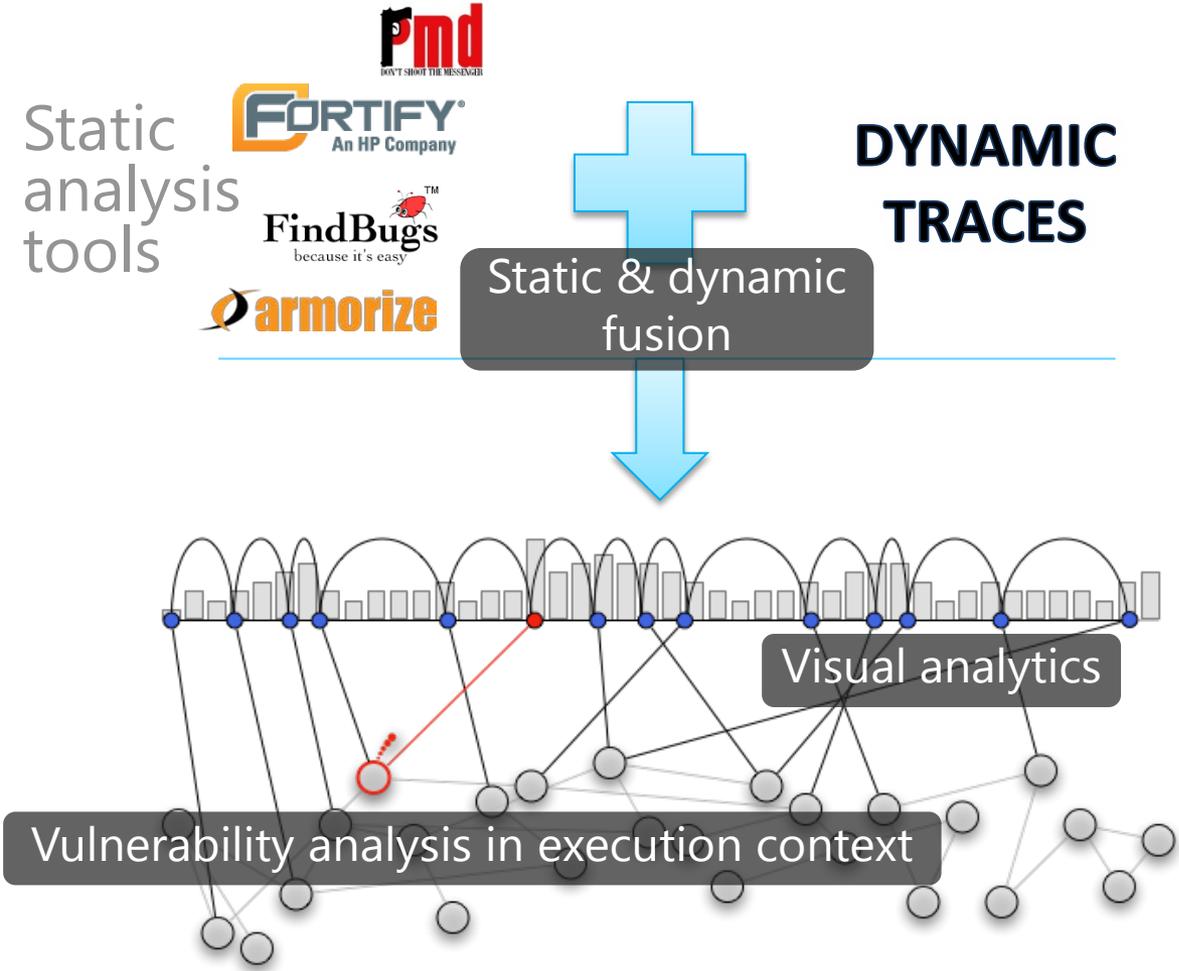
Bridge the gap between security specialists and development teams

3

COMMUNICATION

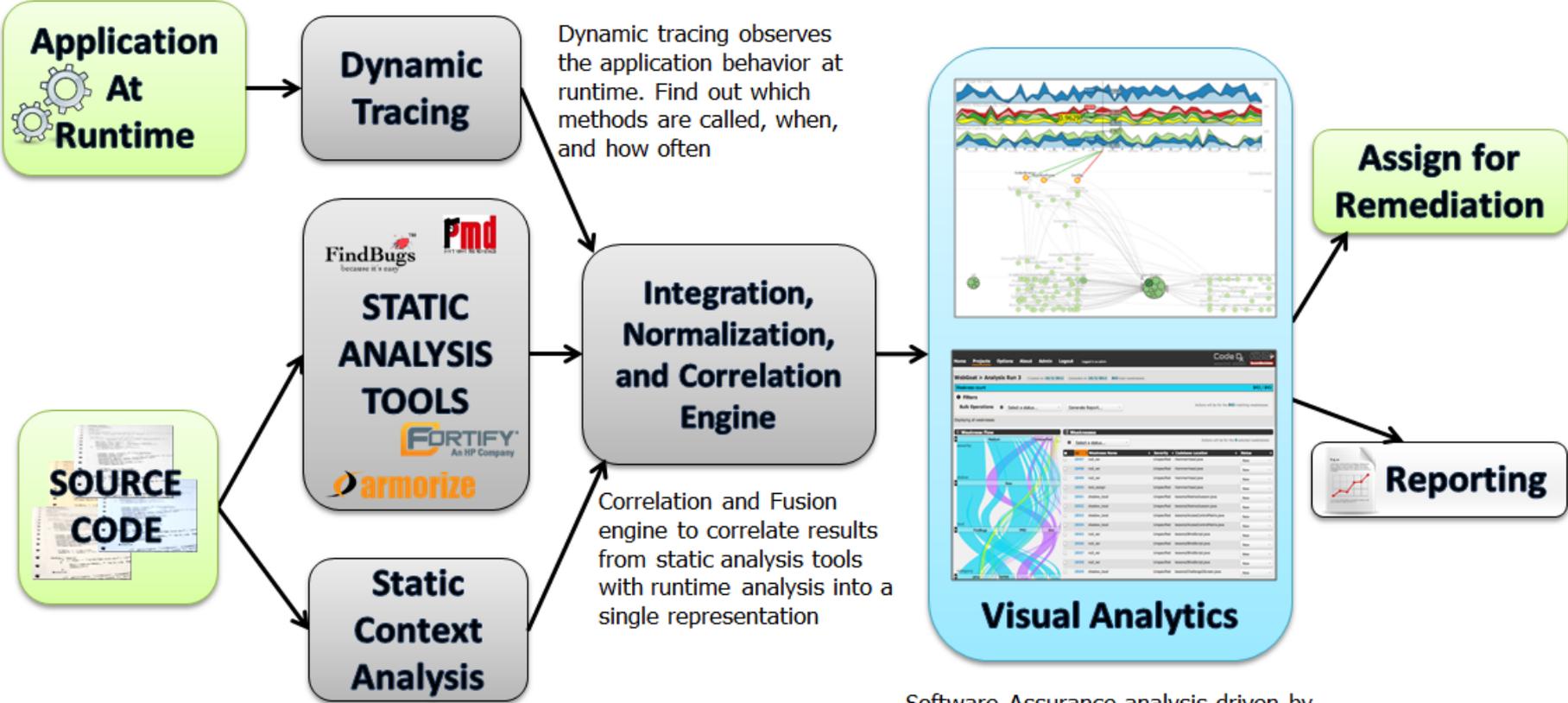


Approach



Dynamic augmented static source analysis

Code Pulse Architecture



- SDLC Integration points
- Automatic, behind-the-scenes processing
- Single interface for software assurance analysis

Software Assurance analysis driven by runtime behavior of application. Users fix weaknesses based on when and how often the affected code was executed.

Use Case Sampling

Dynamic tracing provides two key insights into execution:

- Which code is called
- Call frequency

Interesting use cases:

- Dead code
- Frequently called code
- Critical code paths
- Mature features

Benefits

RUNTIME-DRIVEN ANALYSIS

Scoping analysis to what code is actually executed

AUTOMATION

BEHAVIOR-BASED PRIORITIZATION

Vulnerability prioritization driven by the execution criticality context

PRIORITIZATION

FOCUSED VULNERABILITY PRIORITIZATION

Remove the clutter from the view helping focus on just what matters

VISUAL ANALYSIS

Visual analytics for faster processing

COMMUNICATION

SIMPLIFIED ANALYSIS WORKFLOW

Basing the analysis on execution context bridges the divide between SwA and actual code, helping simplify the workflow

WebGoat > Analysis Run 3 Created on 10/3/2012 Uploaded on 10/3/2012 843 total weaknesses

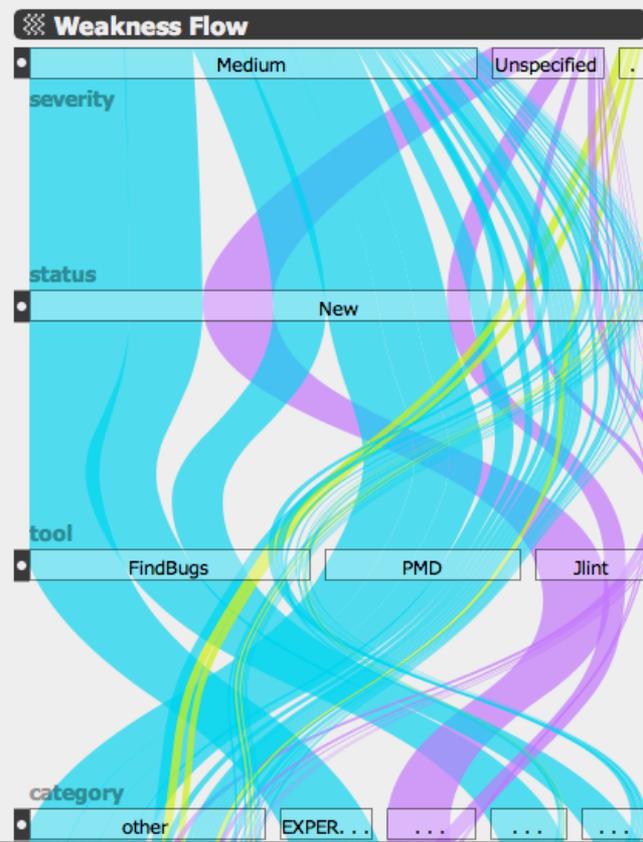


Weakness count 843 / 843

Filters

Bulk Operations ⓘ Actions will be for the 843 matching weaknesses

Displaying all weaknesses



Weaknesses ⓘ Actions will be for the 0 selected weaknesses

<input type="checkbox"/>	Id	Weakness Name	Severity	Codebase Location	Status
<input type="checkbox"/>	28497	null_var	Unspecified	HammerHead.java	New
<input type="checkbox"/>	28498	null_var	Unspecified	HammerHead.java	New
<input type="checkbox"/>	28499	null_var	Unspecified	HammerHead.java	New
<input type="checkbox"/>	28500	lock_assign	Unspecified	HammerHead.java	New
<input type="checkbox"/>	28501	shadow_local	Unspecified	lessons/AbstractLesson.java	New
<input type="checkbox"/>	28502	shadow_local	Unspecified	lessons/AbstractLesson.java	New
<input type="checkbox"/>	28503	shadow_local	Unspecified	lessons/AccessControlMatrix.java	New
<input type="checkbox"/>	28504	shadow_local	Unspecified	lessons/AccessControlMatrix.java	New
<input type="checkbox"/>	28505	null_var	Unspecified	lessons/BlindScript.java	New
<input type="checkbox"/>	28506	null_var	Unspecified	lessons/BlindScript.java	New
<input type="checkbox"/>	28507	null_var	Unspecified	lessons/BlindScript.java	New
<input type="checkbox"/>	28508	null_var	Unspecified	lessons/BlindScript.java	New
<input type="checkbox"/>	28509	shadow_local	Unspecified	lessons/Challenge2Screen.java	New

Challenges

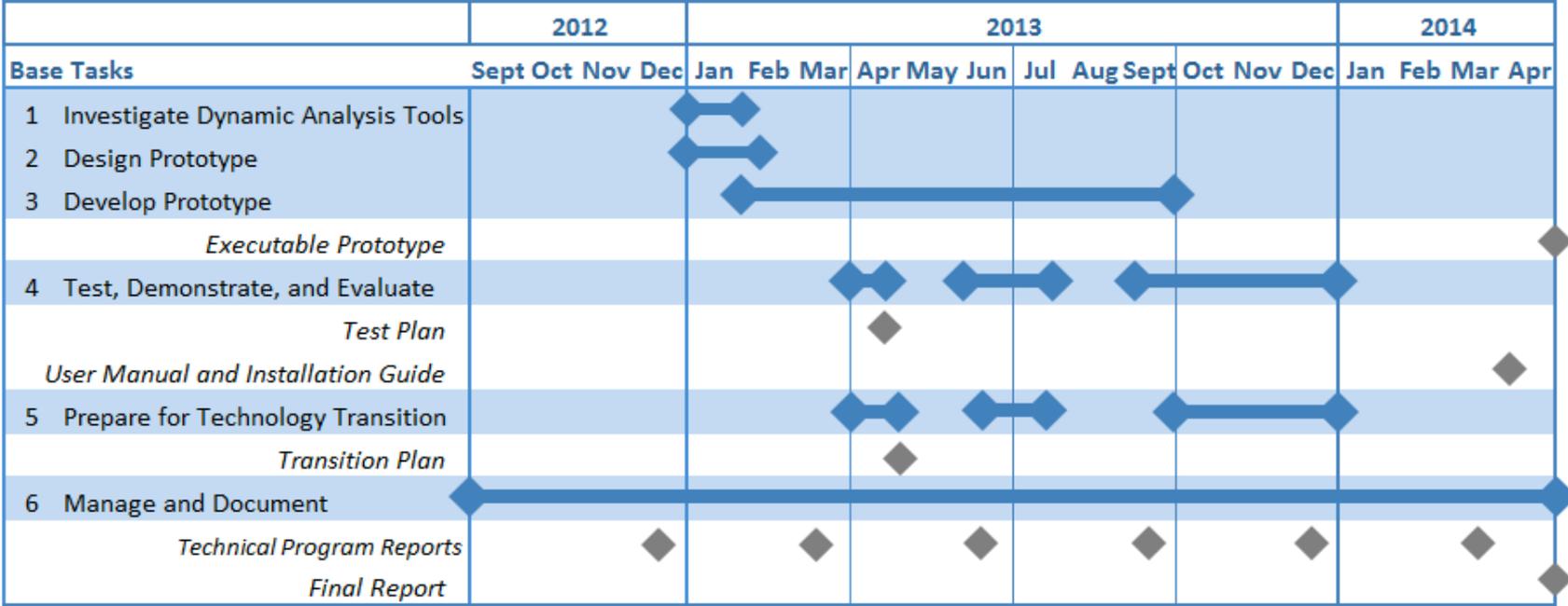
Dynamic Tracing = Lots of data!

- Runtime overhead; need to keep overhead low to make the solution viable
- Data processing & storage challenges; experience dealing with NetFlow and other big data

New visualizations = Potential usability concerns

- Vast visualization repertoire
- Successfully leveraged visualizations in software analytics and other domains
- Early evaluations!

Schedule



Deliverables POP: 8/12 – 4/14		
Mo.	Major Milestones	Deliverables
8	Test plan	Report
8	Transition plan	Report
19	User & Installation Guides	Documents
20	Code Pulse prototype	Software
Quarterly	Technical Reports	Report
20	Final Report	Report

Technology Transition

1. Clarify the value proposition
2. Develop a product roadmap
3. Identify government, commercial, and educational transition partners
4. Develop a marketing strategy
5. Beta testing for assessment and feedback

Technology Transition

Clarify the value proposition

- Conduct interviews with potential users
 - Leverage relationships built in Code Dx
 - Discuss how the tool will help the identified personas
- Present the concept within the community: NCSD's SwA forums

Develop a product roadmap

- Planned features and releases (subject to change)
- Potential service offerings

Technology Transition

Identify transition partners

- SWAMP
- Tool vendors
- Acquisition groups
- Educational institutes

Develop a marketing strategy

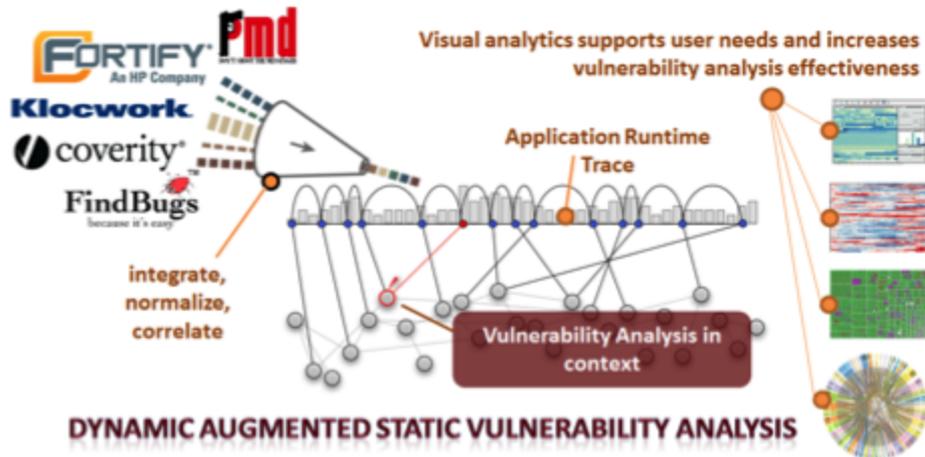
- Product sheets
- Scripted demonstration, video
- Web placement and traffic generation
- Conferences and exhibits
- Articles, both trade and peer-reviewed

Technology Transition

Beta testing for assessment and feedback on:

- Functionality
- Usability
- Robustness
- Performance
- Scalability

Title: Code Pulse: Software Assurance Visual Analytics for Dynamic Analysis of Code



Operational Capability

Performance Targets – A visual software application for context-aware analysis of code that leverages dynamic runtime behavior, correlates disparate software assurance tools, and integrates into the Software development lifecycle (SDLC).

Key Parameters – Results will achieve significant gains in the speed and accuracy of vulnerability analysis, resulting in time-savings and more secure software.

Cost of Ownership – DHS users will receive Code Pulse after completion of program. Maintenance costs are TBD.

BAA Relevance – Will improve vulnerability analysis effectiveness by including dynamic runtime context, and integrate into the SDLC for traceability and remediation.

Proposed Technical Approach

Responds to TTA 1, d, e, (1) – New Tools. Results will improve analysis and remediation of vulnerabilities detected via dynamic runtime analysis of application behavior.

Base Period Tasks

1. Investigate Dynamic Analysis Tools
2. Design Prototype
3. Develop Prototype
4. Test, Demonstrate, and Evaluate
5. Prepare for Technology Transition
6. Manage and Document

Code Pulse will build upon Code Dx, a static-focused correlation and visualization tool developed by Secure Decisions under a Phase II DHS SBIR. Code Pulse work will re-use selected components to save schedule and reduce required funds

Schedule, Cost, Deliverables, & Contact Info

Base Period POP: 8/12 – 4/14

Mo.	Major Milestones	Deliverables
8	Test plan	Report
8	Transition plan	Report
19	User & Installation Guides	Documents
20	Code Pulse prototype	Software
Quarterly	Technical Reports	Report
20	Final Report	Report

Applied Visions, Inc. – Secure Decisions Division

POC: Hassan Radwan

Email: hassan.radwan@SecureDecisions.com

6 Bayview Avenue, Northport, NY 11768 Phone: 518.207.3106

Questions?



Hassan Radwan

Hassan.Radwan@securedisions.com

Ken Prole

Ken.Prole@securedisions.com

Dr. Anita D'Amico

Anita.Damico@securedisions.com

