Release Notice

This presentation is marked

TASC Proprietary

It was approved for Public Release after the notice was applied.

TASC

Defending Software Applications from Threats Through Code Analysis

Glenn D. Fournier
David M. Patterson
Rudy S. Spraycar
Stephen J. Sutton (ret.), (presenter)
Donald M. York
TASC, Inc.

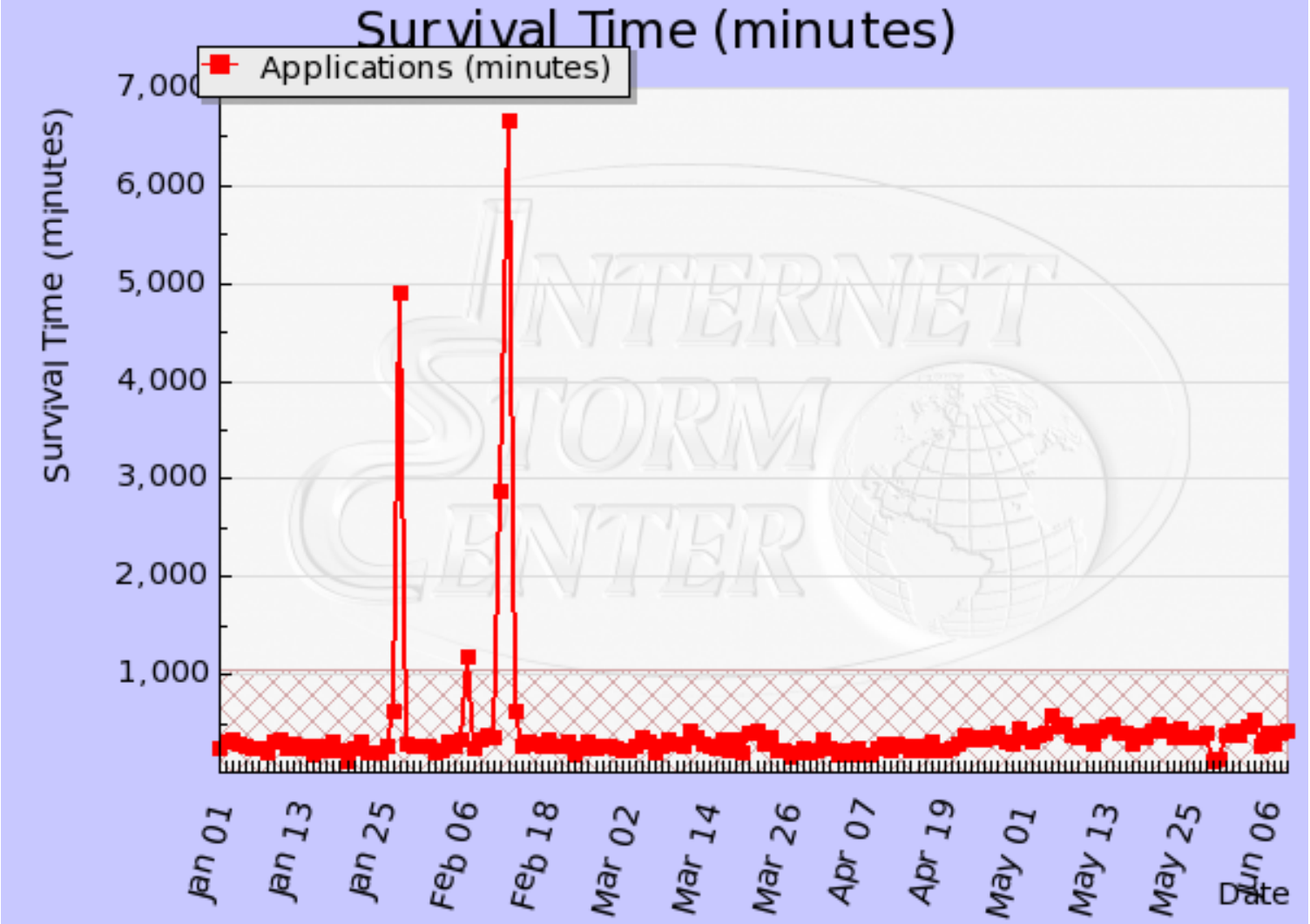
INCOSE Enchantment Chapter
July 13, 2011

Why Code Analysis of Software Applications?

Most Common Attacks

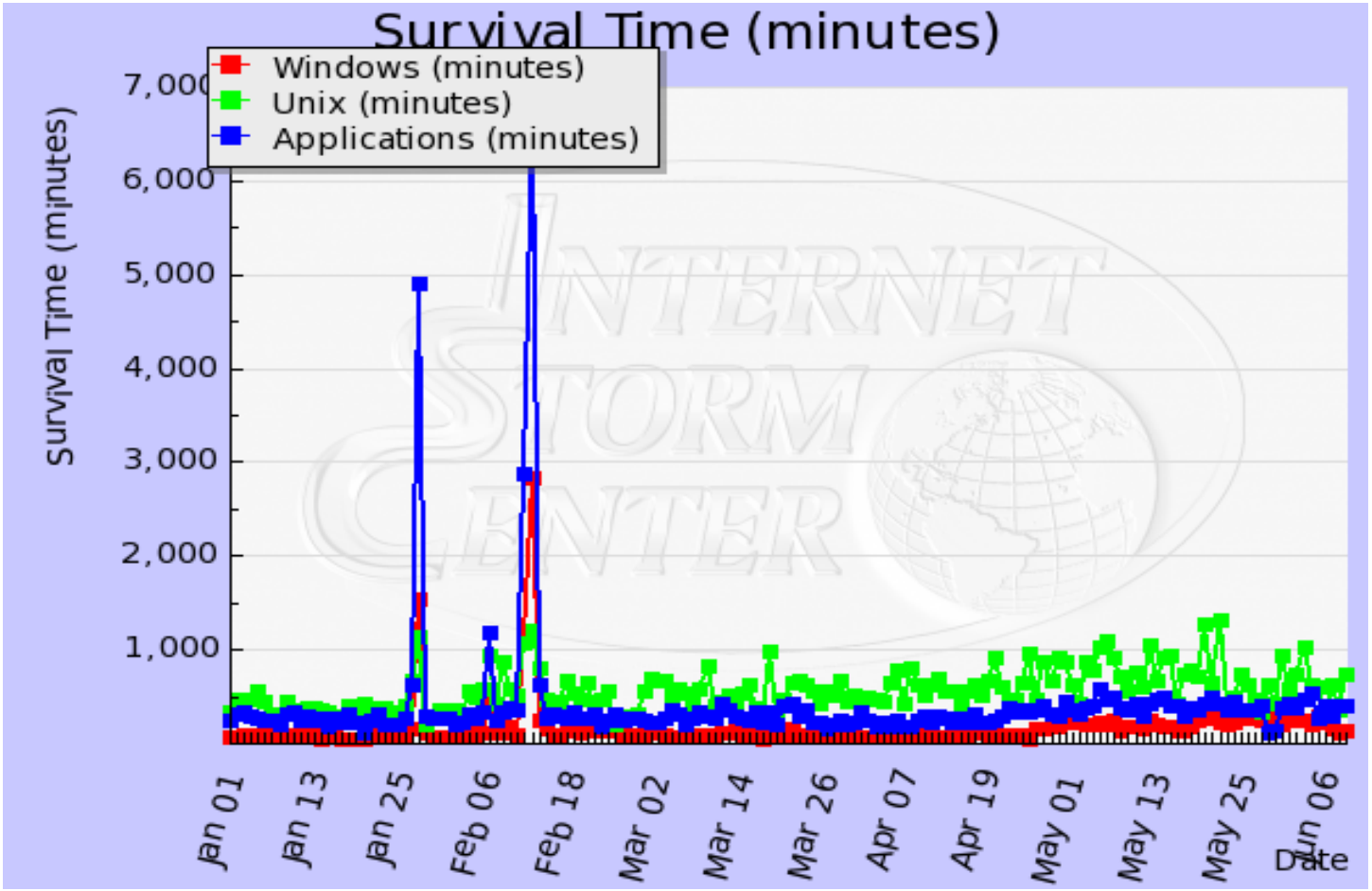
- Acrobat Reader
- Flash Player
- Web Browsers / Apps
- Office Productivity
- Java

How long are your applications safe?



http://isc.sans.org/survivaltime.html

How Long Are you Safe?



TASC

Why Code Analysis of Software Applications?

- *Quality* and *security* of software applications pose risks to mission success
- Risks from process, governance and methodology flaws
  - Concentrating on software function and ignoring software implementation best practices, such as avoiding common weaknesses and vulnerabilities
  - Lack of development discipline and secure code development training
  - Failure to implement processes that adhere to standards and best practice
  - Lack of discipline in executing documented governance
  - Unknown provenance for commercial off the shelf tools
  - Growing code complexity and compounding of vulnerabilities over multiple versions
  - Demand to meet mission timelines

80% or More of Attacks Focus on Applications

Why Code Analysis of Applications? (cont.)

- Demands for assurance in software are growing
  - DCID 6/3 (current policy for near term)
  - ICD 503 (far term policy)
  - NIST 800-53 (expresses requirements; source document for IC and DoD policy)
- Independent, objective assessment
  - Identify security and quality vulnerabilities before deployment
  - Identify weaknesses before an adversary or circumstances can exploit them
  - Generate lessons learned
  - Stimulate process improvement

Can't Eliminate All Vulnerabilities, but Can Improve Security Over Time

Benefits of Software Application Code Analysis

Key Benefits of Application Code Analysis

- Provides actionable information for owners of software applications
  - Operational Risk Assessment
    - For acceptable risk level
      - Implement operational environment mitigations
      - Monitor environment for events related to software risks
    - For unacceptable risk level
      - Require mitigations by development team
      - Identify alternative software solution
  - Evaluate Developer Team
    - Security best practices
    - Code quality
- Promotes an environment for the continued reduction of operational risk

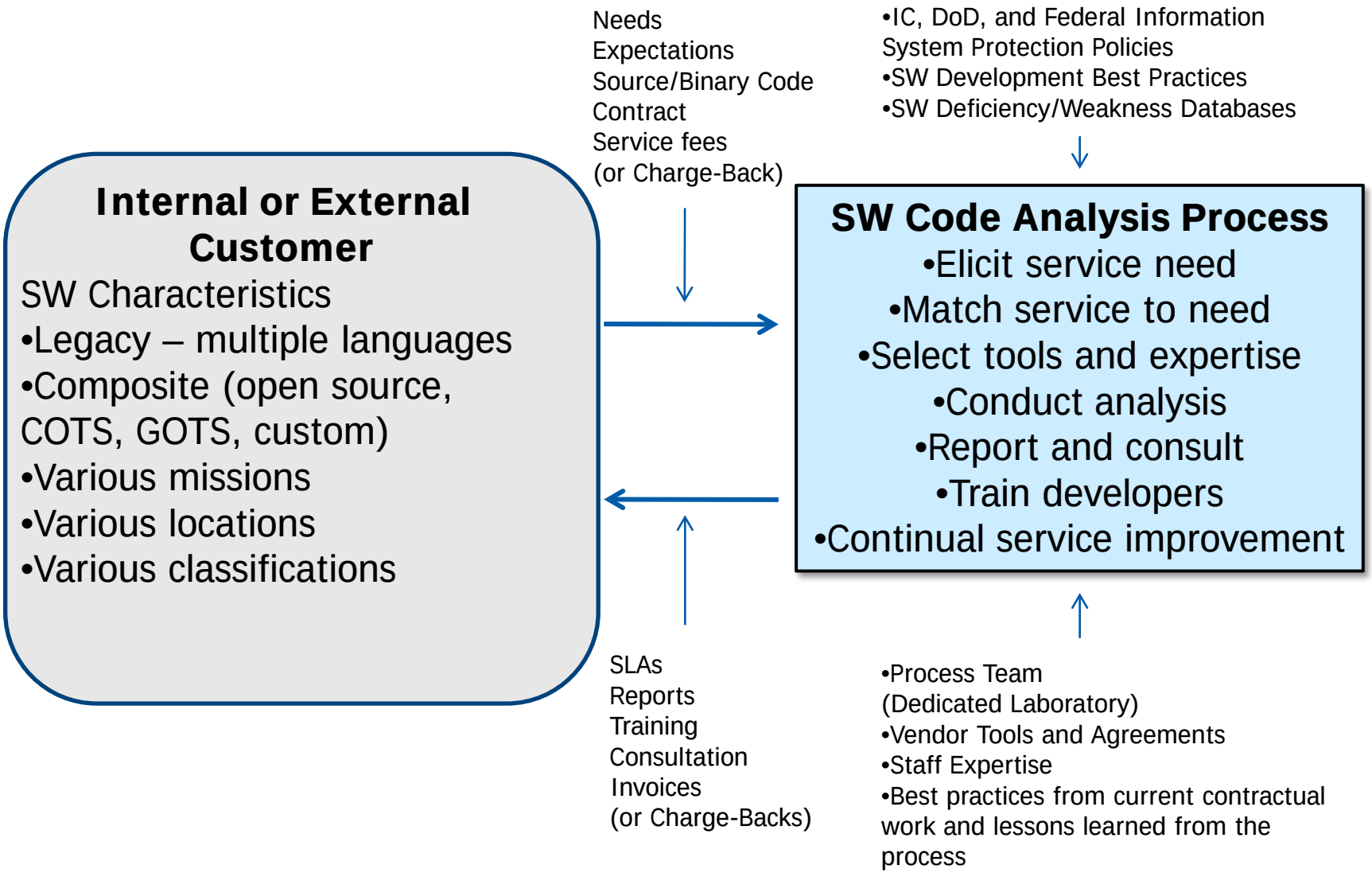
Benefits of Code Analysis for Agile Development

- Technology debt[1] – Agile development teams deliver code that includes bugs, design issues, and other code quality problems that are potentially introduced with every addition or change to the code
  - If steps are not taken to minimize technical debt, change can become prohibitively expensive, making new capabilities unresponsive to new customer requirements.
- Code analysis employed during the initial spin of development and then applied with every additional spin can help alleviate the accumulation of errors and associated vulnerabilities

[1]Black, Sue; Boca, Paul P; et al. September 2009. *Formal Versus Agile: Survival of the Fittest?* Computer. IEEE Computer Society. New York, NY. p. 44.

A Plan for a Code Analysis Process

Code Analysis Process: Concept of Operations



An Analysis of Sample Code

Sample Code Analysis

- Sample of Open Source Embedded Code
  - SOA framework for instantiating an application through a GUI; has initial set of components
  - Code written in C/C++, Python
  - Scripts written using XML
- Analysis Approach
  - Define an unclassified use case
  - Define and identify a risk taxonomy to judge the findings
  - Scan code with vendor and open source tools
  - Manually verify tool results; manually review major code elements

Risk Determination

LIKELIHOOD				
	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Informational	**Low**	Medium
		Low	**Medium**	**High**
		IMPACT		

Critical	Critical issue. Resolve immediately. Easy to discover/exploit. High value assets/capabilities.
High	Significant security issue. Needs attention. May be hard to discover/exploit or not involve high value assets/capabilities.
Medium	Potential security issue. Address in future release. Issues may not currently be exploitable but could become so.
Low	Minimal security risk in likelihood/consequence. Issues with low likelihood/impact.
Informational	Note good/bad practices, unsuccessful attempts to penetrate the system, or other—not a security issue.

Open Source Results

- Open Source Tool #1 – Identified 369 Total findings
  - 38 Total Unique Findings
    - Risk Level 5 – 8 Total Findings
    - Risk Level 4 – 87 Total Findings
    - Risk Level 3 – 4 Total Findings
    - Risk Level 2 – 161 Total Findings
    - Risk Level 1 – 109 Total Findings
- Open Source Tool #2– Identified 424 Total Findings
  - 26 Total Unique Findings
    - Risk Level High – 76
    - Risk Level Medium – 54
    - Risk Level Low – 294

Sample Code Analysis Findings

- Automated Analysis
  - Total findings identified
    - Commercial Tool – 335
    - Open Source Tool #1– 369
    - Open Source Tool #2 – 424
- Manual Analysis
  - Reduced findings to 23 total findings
    - 22 identified first through automated tools
    - 1 identified through manual analysis alone
  - Eliminated false positives and unexploitable code quality findings, such as:
    - Memory Leaks
    - Poor Style
    - Dead Code

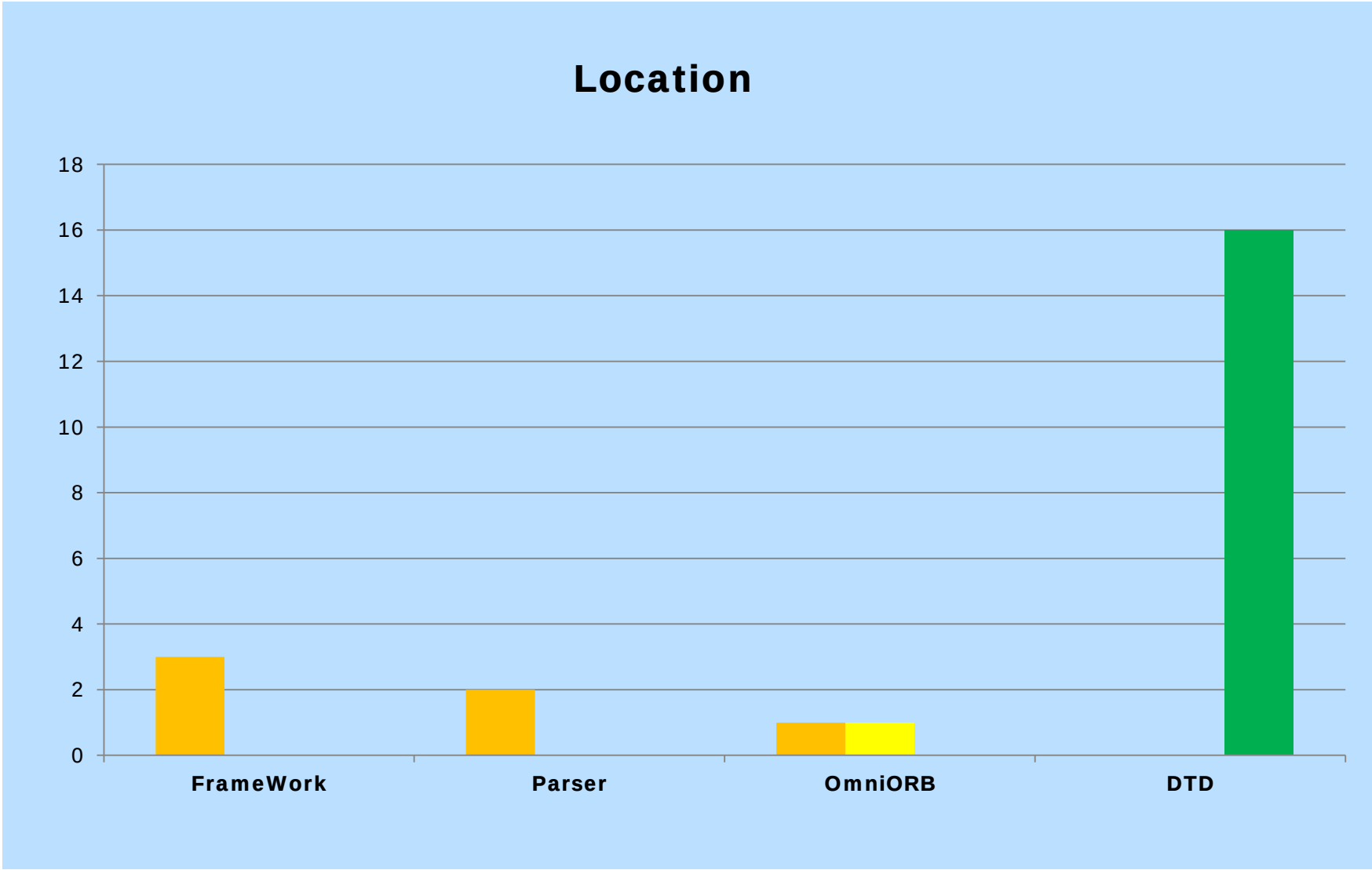
Sample Code Analysis Findings (cont.)

- Significant vulnerabilities found (number of instances)
  - Attackers Can Access CORBA Objects Due To Lack of Authentication (1 - manual)
  - Attacker Can Cause Buffer Overflow Due to Unverified Bounds (2)
  - Attackers With System Access Can Execute Arbitrary Commands Through Directory Traversal (3)
  - Attackers Can Cause Application Errors Due to Weak Input Validation Scheme (16)
  - Attackers Can Eavesdrop on System Communications Due to Lack of SSL (1)

2010 CWE/SANS Top 25 Most Dangerous Errors Leading to Vulnerabilities[1]

1 CWE-79 Failure to Preserve Web Page Structure ('Cross-site Scripting')	**10 CWE-311** Missing Encryption of Sensitive Data	**19 CWE-306** Missing Authentication for Critical Function
2 CWE-89 Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')	**11CWE-**798 Use of Hard-coded Credentials	**20 CWE-**494 Download of Code Without Integrity Check
3 CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	**12 CWE-**805 Buffer Access with Incorrect Length Value	**21 CWE-**732 Incorrect Permission Assignment for Critical Resource
4 CWE-352 Cross-Site Request Forgery (CSRF)	**13 CWE-**98 Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	**22 CWE-**770 Allocation of Resources Without Limits or Throttling
5 CWE-285 Improper Access Control (Authorization)	**14 CWE-**129 Improper Validation of Array Index	**23 CWE-**601 URL Redirection to Untrusted Site ('Open Redirect')
6 CWE-807 Reliance on Untrusted Inputs in a Security Decision	**15 CWE-**754 Improper Check for Unusual or Exceptional Conditions	**24 CWE-**327 Use of a Broken or Risky Cryptographic Algorithm
7 CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	**16 CWE-**209 Information Exposure Through an Error Message	**25 CWE-**362 Race Condition
8 CWE-434 Unrestricted Upload of File with Dangerous Type	**17 CWE-**190 Integer Overflow or Wraparound	
9 CWE-78 Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')	**18 CWE-**131 Incorrect Calculation of Buffer Size	[1]http://cwe.mitre.org/top25/

Findings of Assessment by Location in Sample



A Close Look at One Vulnerability

- Attackers Can Access CORBA Objects Due to Lack of Authentication
  - Location: omniORB Service
  - Severity: Medium
  - Risk: The framework does not provide a means by which interacting components can authenticate one another through the CORBA service. *The current CORBA implementation does not utilize authentication or authorization, so any remote CORBA objects can be exposed to an attacker.* An attacker can set up a rogue communications endpoint to access or modify objects via the CORBA service. Without proper authentication, framework components would be unable to differentiate legitimate data and commands placed in the naming service from injected or forged ones. As a result, the application cannot provide proper accountability or access control. If this system is used in a closed environment, risk is LOW, but should an attack occur the impact is nevertheless HIGH.

A Close Look at One Vulnerability (cont.)

- Recommended Solution: Implement an authentication mechanism to verify the identity of users and/or distributed components. Alternative methods:
  - The Common Secure Interoperability, version 2 for CORBA implements client/server authentication at the transport layer or above. This standard also provides transport encryption and integrity as well as other security services using CORBA. However, the CORBA implementation used by the this application, omniORB, does not support Common Secure Interoperability, version 2 (CSIv2).
  - Implement a custom authentication mechanism that operates in conjunction with this framework. The mechanism should protect any identity credentials at all times and should be resistant to attack. The mechanism may also need to provide management functions to configure allowed credentials.
  - Implement a network-based scheme using a combination of access control lists and network policies. This scheme would segment the network such that only legitimate systems could communicate with each other. A certificate-based authentication mechanism could be used to reinforce network identity. Note that this scheme does not prevent a malicious user on one system from connecting to other endpoints in the network.

Code Analysis Tools – Strengths & Weaknesses

Strengths of Automated Tools

- The Pros…
  - Analyze large amounts of code in a short period of time
  - Allow developers to maintain the development tempo
  - Reduce the work to manually address ~20% of vulnerabilities
- Commercial tools…
  - Are continually improved and updated by vendors
  - Provide a variety of reporting options
    - Summary
    - Executive
    - Detailed
  - Ability to drill down into findings
  - References and/or links for remediation

Tools provide a necessary level of automation for the process

Automated Tools are not Enough

- The Cons…
  - Provide an excessive amount of data
  - Do not consider intent of code, threat environment, or areas of specific customer concern
  - May miss serious vulnerabilities
- Utilizing multiple tools provides a more comprehensive description of operational risk
- Expertise and manual review are necessary to overcome these weaknesses while maintaining the benefits of automated tools

However, tools are not sufficient to give a complete assessment

Expertise and Manual Review

Expertise and Manual Review

- Employ the expertise of…
  - Senior Software Development Engineers
  - Senior Security Engineers
- … to conduct a manual review of the software and automated tool findings
  - Applied to automated tool findings
    - Minimizes false positives by validating automated tool findings
    - Focuses remediation efforts on findings that are not mitigated by other environmental factors
  - Applied directly to code base
    - Minimizes false negatives by identifying findings missed by automated tools
    - Focuses analysis efforts on areas of customer importance that could be missed by automated tools

Use Expertise and Manual Review to Overcome the Weaknesses but Maintain the Benefits of Automated Tools

QUESTIONS?

What this Process Does

- The Software Code Analysis Process Should…
  - Identify significant security vulnerabilities in applications
  - Utilize static and dynamic analysis techniques
  - Consider the perspective of a malicious user while conducting analysis activities
  - Isolate findings to specific locations or sub-components within the software application
  - Offer mitigations for identified vulnerabilities

Process Description

- The software code review process provides a comprehensive review of source code and/or binary code for flaws, vulnerabilities, backdoors and exploitable errors through Static Analysis
- Along with a review of the code, the review team should identify potential ways an adversary could exploit the application from a running state
  - (e.g., SQL injections from a Web front end, unhandled errors, etc.)
- Mitigation information, prioritized, should be provided to the development team, allowing them to decide on actions related to enhancing the security of and reducing the risks to the application in question
- The objective is not just a simple code review, but a review from the point of view of the adversary, to see how an adversary can exploit an application, and then prevent this from taking place

Improving the Process at TASC

- Drive for continual service improvement in results, products and assessment thoroughness, expanded capabilities, and staff knowledge and expertise
- Determination to enhance the assurance of Customer applications (and systems)
- Best practices from project- and organization-specific experience, and lessons learned in the laboratory