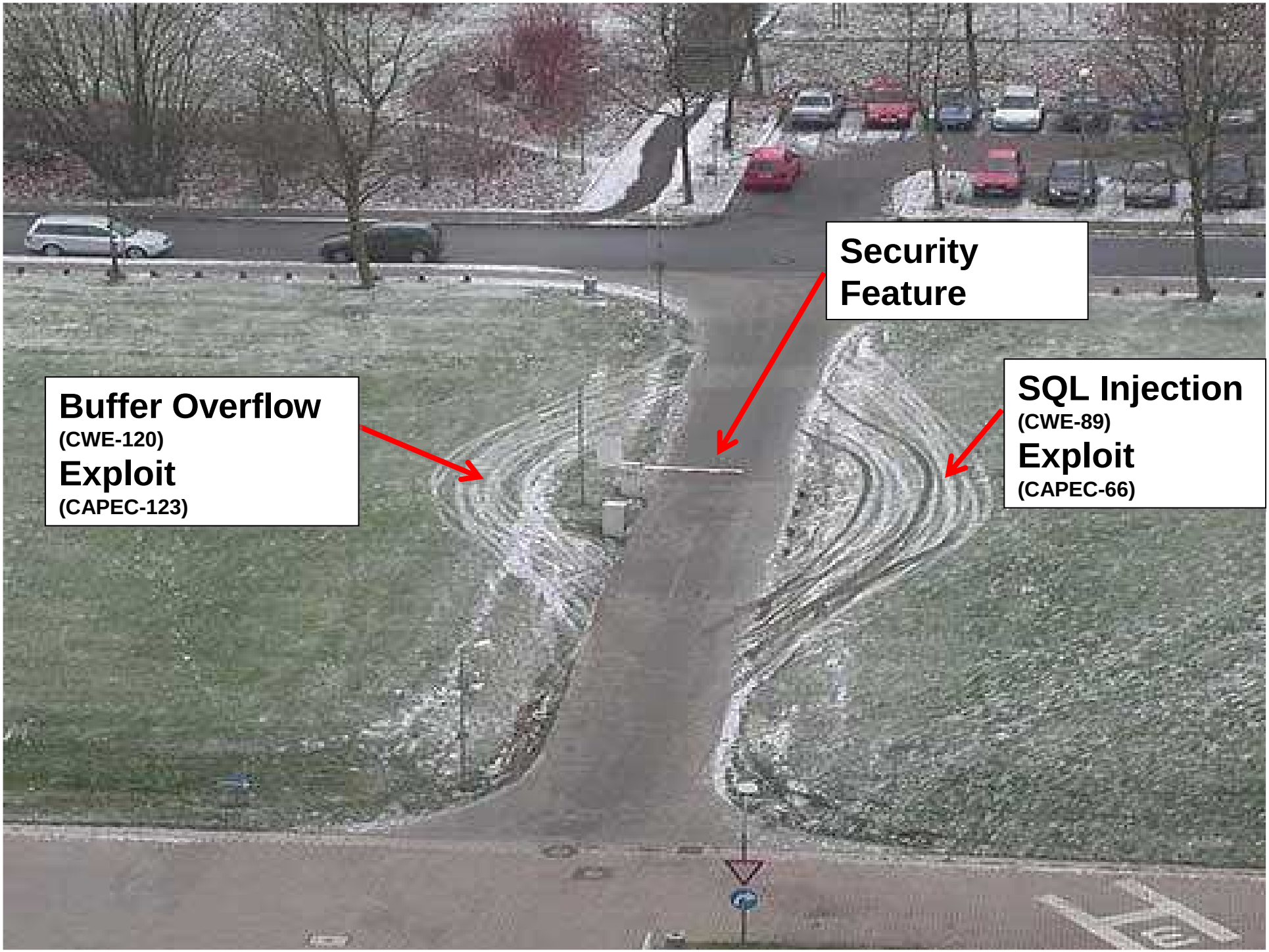
SCAP and SwA Automation Protocol

Bob Martin

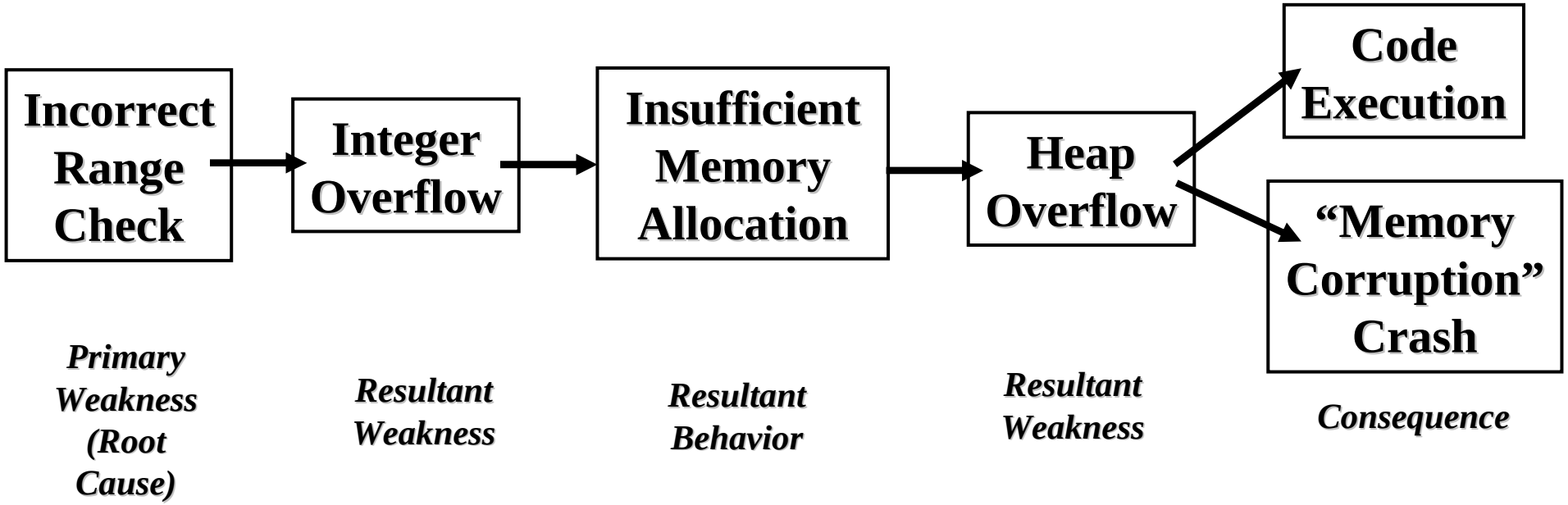
June 21, 2010

Homeland Security


Missing Authentication for Critical Function (CWE-306)
Using Unpublished Web Service APIs (CAPEC-36)

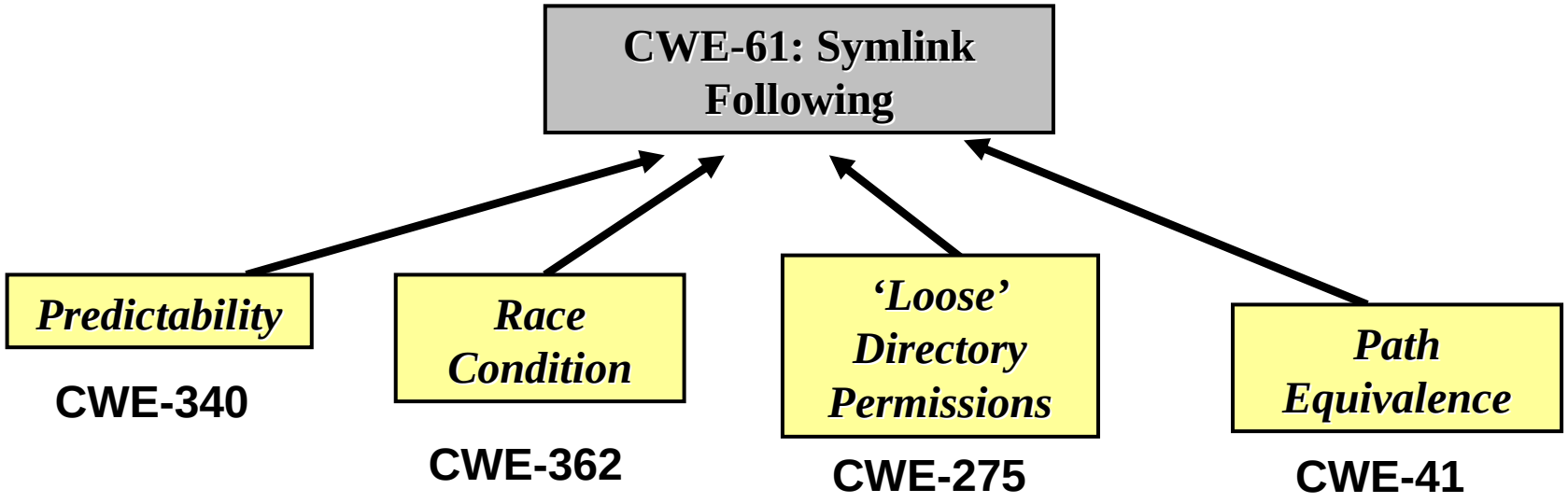

Security Feature
Buffer Overflow
(CWE-120)
Exploit
(CAPEC-123)
SQL Injection
(CWE-89)
Exploit
(CAPEC-66)

Many Vulnerabilities are Multi-Factor: Chains and Composites



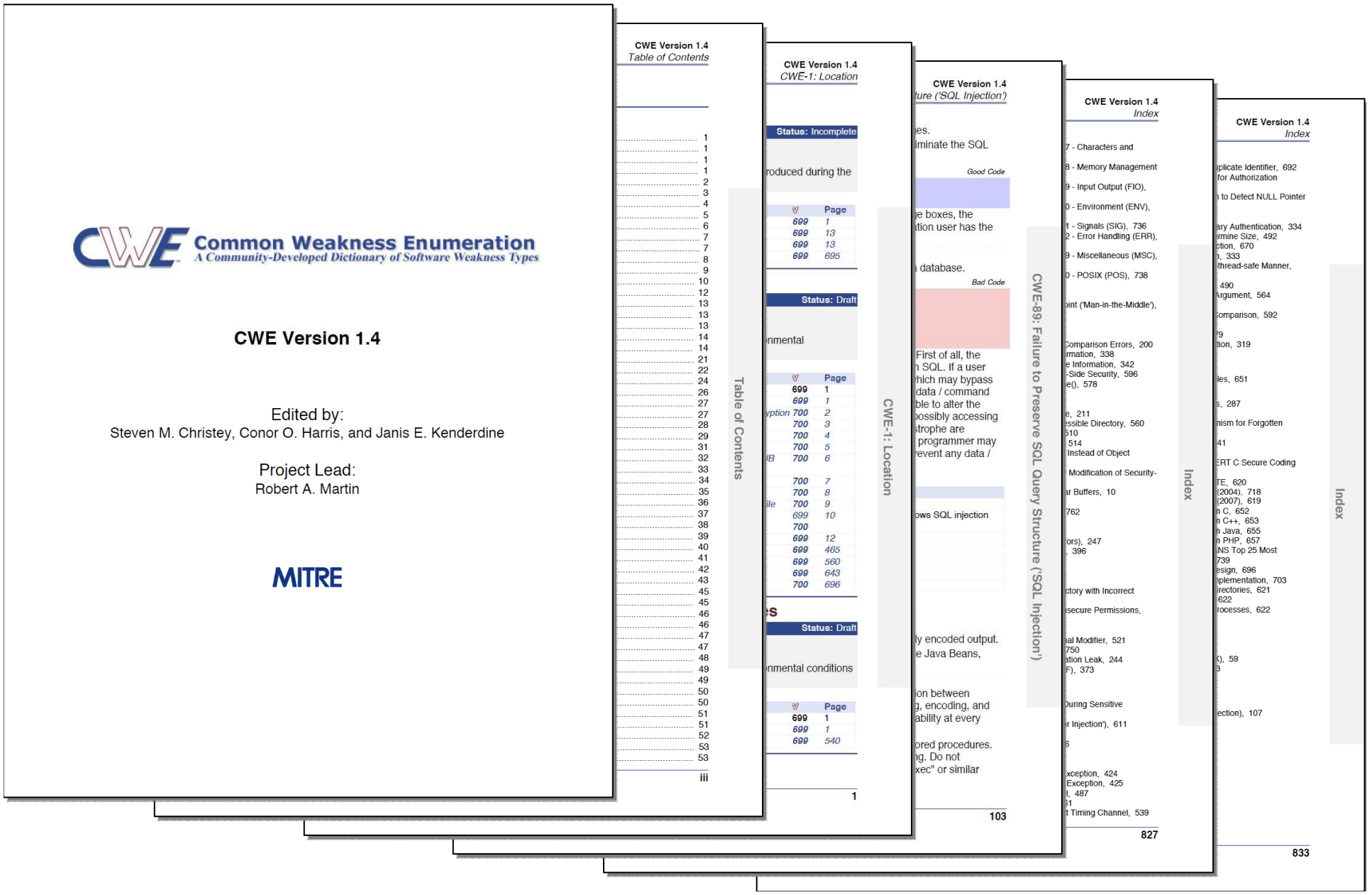
One of the main problems with classification and terminology is that ANY behavior in the chain could be regarded as the vulnerability.

Composite: Symbolic Link Following



- Filename can be predicted
- File can be created by other party before it is opened for writing
- File created in a shared directory with writable permissions
- Equivalence: a symlink can act an alternate name for a critical file

Printable PDFs of Entire CWE Now Available



CWE Outreach: A Team Sport

May/June Issue of IEEE Security & Privacy…

SANS Common Security Errors in Programming

Special thanks to Robert A. Martin of MITRE Corporation.

The SANS Common Security Errors in Programming map illustrates the software weaknesses that are responsible for the majority of the publicly known vulnerabilities discovered in 2006. It is based on the CWE (Common Weakness Enumeration) that provides a unified, measurable set of software weaknesses that will enable more effective discussion and actions to find these weaknesses in source code and eliminate them. The CWE was developed by MITRE and sponsored by the Department of Homeland Security. The numbers between parentheses represent the common weakness enumeration IDs for each weakness. Numbers between square brackets are direct children of the weakness listed. CWE IDs can be found at the MITRE CWE Website or accessed directly by putting the number (in place of ###) in the following URL:

http://cwe.mitre.org/data/definitions/###.html

SANS MITRE

2009's Top 25 CVE Causes and Important CWEs

Handler Errors

- Deployment of Wrong Handler
- Missing Handler
- Dangerous Handler not Disabled During Sensitive Operations
- Unparsed Raw Web Content Delivery
- Incomplete Identification of Uploaded File Variables (PHP)
- Unrestricted File Upload

User Interface Errors

- UI Discrepancy for Security Feature
- Multiple Interpretations of UI Input
- UI Misrepresentation of Critical Information

Behavioral Problems

- Behavioral Change in New Version or Environment
- Expected Behavior Violation

Initialization and Cleanup Errors

- Insecure Default Variable Initialization
- External Initialization of Trusted Variables
- Non-exit on Failed Initialization
- Missing Initialization
- Incomplete Cleanup
- Improper Cleanup on Thrown Exception
- Improper Initialization - (665)

Channel and Path Errors

- Channel Errors
- Failure to Protect Alternate Path
- Uncontrolled Search Path Element
- Unquoted Search Path or Element
- Untrusted Search Path

Error Handling

- Error Conditions, Return Values, Status Codes
- Failure to Use a Standardized Error Handling Mechanism
- Failure to Catch All Exceptions in Servlet
- Not Failing Securely ('Failing Open')
- Missing Custom Error Page

Pointer Issues

- Return of Pointer Value Outside of Expected Range
- Use of size of() on a Pointer Type
- Incorrect Pointer Scaling
- Use of Pointer Subtraction to Determine Size
- Assignment of a Fixed Address to a Pointer
- Attempt to Access Child of a Non-structure Pointer

Time and State

- State Issues
  - Incomplete Internal State Distinction
  - State Synchronization Error
  - Mutable Objects Passed by Reference
  - Passing Mutable Objects to an Untrusted Method
  - External Control of Critical State Data - (642)
- Race Condition - (362)
- Session Fixation
- Concurrency Issues
- Temporary File Issues
- Covert Timing Channel
- Technology-Specific Time and State Issues
- Symbolic Name not Mapping to Correct Object
- Signal Errors
- Unrestricted Externally Accessible Lock
- Double-Checked Locking
- Insufficient Session Expiration
- Insufficient Synchronization
- Use of a Non-reentrant Function in an Unsynchronized Context
- Improper Control of a Resource Through its Lifetime
- Exposure of Resource to Wrong Sphere
- Incorrect Resource Transfer Between Spheres
- Use of a Resource after Expiration or Release
- External Influence of Sphere Definition
- Uncontrolled Recursion
- Redirect Without Exit

Failure to Fulfill API Contract ('API Abuse')

- Failure to Clear Heap Memory Before Release ('Heap Inspection')
- Call to Non-ubiquitous API
- Use of Inherently Dangerous Function
- Multiple Binds to the Same Port
- J2EE Bad Practices: Direct Management of Connections
- Incorrect Check of Function Return Value
- Often Misused: Arguments and Parameters
- Uncaught Exception
- Execution with Unnecessary Privileges - (250)
- Often Misused: String Management
- J2EE Bad Practices: Direct Use of Sockets
- Unchecked Return Value
- Failure to Change Working Directory in chroot Jail
- Reliance on DNS Lookups in a Security Decision
- Failure to Follow Specification
- Failure to Provide Specified Functionality

Web Problems

- Failure to Sanitize CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
- Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
- Improper Sanitization of HTTP Headers for Scripting Syntax
- Use of Non-Canonical URL Paths for Authorization Decisions

Indicator of Poor Code Quality

- NULL Pointer Dereference
- Incorrect Block Delimitation
- Omitted Break Statement in Switch
- Undefined Behavior for Input to API
- Use of Hard-coded, Security-relevant Constants
- Unsafe Function Call from a Signal Handler
- Suspicious Comment
- Return of Stack Variable Address
- Missing Default Case in Switch Statement
- Expression Issues
- Use of Obsolete Functions
- Use of Function with Inconsistent Implementations
- Unused Variable
- Dead Code
- Resource Management Errors
- Improper Resource Shutdown or Release - (404)
- Empty Synchronized Block
- Explicit Call to Finalize()
- Reachable Assertion
- Use of Potentially Dangerous Function

Data Handling

- Numeric Errors
  - Use of Incorrect Byte Ordering
  - Unchecked Array Indexing
  - Incorrect Conversion between Numeric Types
    - Unexpected Sign Extension
    - Signed to Unsigned Conversion Error
    - Unsigned to Signed Conversion Error
    - Numeric Truncation Error
  - Incorrect Calculation - (682)
    - Incorrect Calculation of Buffer Size
    - Integer Overflow or Wraparound
    - Integer Underflow (Wrap or Wraparound)
    - Off-by-one Error
    - Divide By Zero
- Representation Errors
  - Cleansing, Canonicalization, and Comparison Errors
  - Reliance on Data/Memory Layout
- Information Management Errors
  - Information Leak (Information Disclosure)
    - Information Leak Through Sent Data
    - Privacy Leak through Data Queries
    - Discrepancy Information Leaks
    - Error Message Information Leak - (209)
    - Cross-Boundary Cleansing Information Leak
    - Intended Information Leak
    - Process Environment Information Leak
    - Information Leak Through Debug Information
    - Sensitive Information Uncleared Before Release
    - Information Leak of System Data
    - Information Leak Through Caching
    - Information Leak Through Environmental Variables
    - File and Directory Information Leaks
    - Information Leak Through Query Strings in GET Request
    - Information Leak Through Indexing of Private Data
  - Information Loss or Omission
  - Containment Errors (Container Errors)
- Improper Access of Indexable Resource ('Range Error')
- Type Errors
- Improper Encoding or Escaping of Output - (116)
- String Errors
- Data Structure Issues
- Improper Handling of Syntactically Invalid Structure
- Modification of Assumed-Immutable Data (MAID)
- Improper Input Validation - (20)
  - Pathname Traversal and Equivalence Errors
  - Process Control
  - Missing XML Validation
  - Failure to Sanitize Data into a Different Plane ('Injection')
    - Improper Sanitization of Special Elements used in a Command ('Command Injection') - (77)
    - Failure to Preserve Web Page Structure ('Cross-site Scripting') - (79)
    - Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection') - (89)
    - Failure to Sanitize Data into LDAP Queries ('LDAP Injection')
    - XML Injection (aka Blind XPath Injection)
    - Failure to Sanitize CRLF Sequences ('CRLF Injection')
    - Uncontrolled Format String
    - Failure to Sanitize Special Elements into a Different Plane
      - Argument Injection or Modification
      - Improper Control of Resource Identifiers ('Resource Injection')
    - Failure to Control Generation of Code ('Code Injection') - (94)
    - Improper Sanitization of Special Elements
  - Technology-Specific Input Validation Problems
  - Misinterpretation of Input
  - Unchecked Input for Loop Condition
  - Null Byte Interaction Error (Poison Null Byte)
  - Direct Use of Unsafe JNI
  - Improper Output Sanitization for Logs
  - Failure to Constrain Operations within the Bounds of a Memory Buffer - (119)
  - Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
  - ASP.NET Misconfiguration: Not Using Input Validation Framework
  - URL Redirection to Untrusted Site ('Open Redirect')
  - Variable Extraction Error
  - Unvalidated Function Hook Arguments
  - External Control of File Name or Path - (73)
  - Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code
  - Use of Path Manipulation Function without Maximum-sized Buffer

Security Features

- Credentials Management
  - Hard-Coded Password - (259)
  - Unverified Password Change
  - Missing Password Field Masking
  - Weak Cryptography for Passwords
  - Weak Password Requirements
  - Not Using Password Aging
  - Password Aging with Long Expiration
  - Insufficiently Protected Credentials
  - Weak Password Recovery Mechanism for Forgotten Password
- Insufficient Verification of Data Authenticity
  - Origin Validation Error
  - Improper Verification of Cryptographic Signature
  - Use of Less Trusted Source
  - Acceptance of Extraneous Untrusted Data With Trusted Data
  - Improperly Trusted Reverse DNS
  - Insufficient Type Distinction
  - Cross-Site Request Forgery (CSRF) - (352)
  - Failure to Add Integrity Check Value
  - Improper Validation of Integrity Check Value
  - Trust of System Event Data
  - Reliance on File Name or Extension of Externally-Supplied File
  - Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
- Privacy Violation
- Reliance on Cookies without Validation and Integrity Checking
- Client-Side Enforcement of Server-Side Security - (602)
- Improperly Implemented Security Check for Standard
- Improper Authentication
- User Interface Security Issues
- Use of Insufficiently Random Values - (330)
- Logging of Excessive Data
- Certificate Issues
- Cryptographic Issues
  - Key Management Errors
  - Missing Required Cryptographic Step
  - Not Using a Random IV with CBC Mode
  - Failure to Encrypt Sensitive Data
    - Cleartext Storage of Sensitive Information
    - Cleartext Transmission of Sensitive Information - (319)
    - Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
  - Reversible One-Way Hash
  - Inadequate Encryption Strength
    - Use of a Broken or Risky Cryptographic Algorithm - (327)
  - Use of RSA Algorithm without OAEP
- Permissions, Privileges, and Access Controls
  - Access Control (Authorization) Issues - (285)
  - Permission Issues
    - Incorrect Default Permissions
    - Insecure Inherited Permissions
    - Insecure Preserved Inherited Permissions
    - Incorrect Execution-Assigned Permissions
    - Improper Handling of Insufficient Permissions or Privileges
    - Improper Preservation of Permissions
    - Exposed Unsafe ActiveX Method
    - Incorrect Permission Assignment for Critical Resource - (732)
    - Permission Race Condition During Resource Copy
  - Privilege / Sandbox Issues
  - Improper Ownership Management
  - Insecure User Management
- Password in Configuration File
- Insufficient Compartmentalization
- Reliance on a Single Factor in a Security Decision
- Insufficient Psychological Acceptability
- Reliance on Security through Obscurity
- Protection Mechanism Failure
- Insufficient Logging
- Reliance on Cookies without Validation and Integrity Checking in a Security Decision

Insufficient Encapsulation

- Mobile Code Issues/Missing Custom Error Page
  - Public cloneable() Method Without Final ('Object Hijack')
  - Use of Inner Class Containing Sensitive Data
  - Critical Public Variable Without Final Modifier
  - Download of Code Without Integrity Check - (494)
  - Array Declared Public, Final, and Static
  - Finalize() Method Declared Public
- Leftover Debug Code
- Use of Dynamic Class Loading
- clone() Method Without super.clone()
- Comparison of Classes by Name
- Data Leak Between Sessions
- Trust Boundary Violation
- Reliance on Package-level Scope
- J2EE Framework: Saving Unserializable Objects to Disk
- Deserialization of Untrusted Data
- Serializable Class Containing Sensitive Data
- Information Leak through Class Cloning
- Public Data Assigned to Private Array-Typed Field
- Private Array-Typed Field Returned From A Public Method
- Public Static Final Field References Mutable Object
- Exposed Dangerous Method or Function
- Critical Variable Declared Public
- Access to Critical Private Variable via Public Method

OWASP Top Ten 2007 & 2010 use CWE refs

OWASP TOP 10

THE TEN MOST CR
APPLICATION SECU

2007 UPDATE

Our methodology for the Top 10 2007 was simple: take the MITRE Vulnerability Trends for 2006, and distill the Top 10 *web application security* issues. The ranked results are as follows:

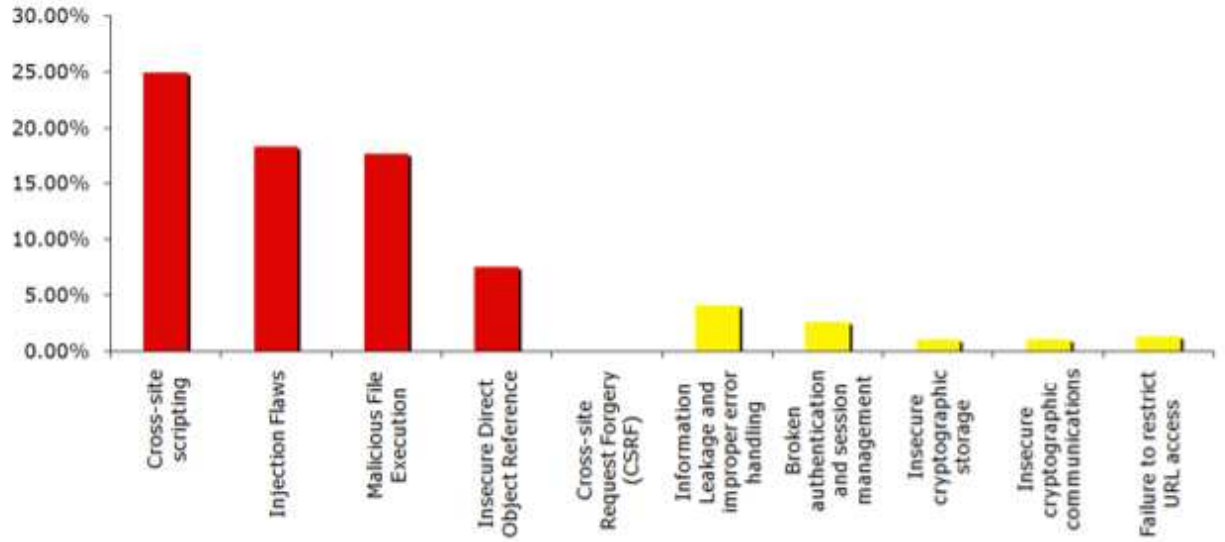
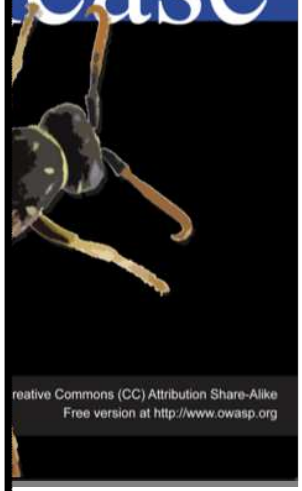


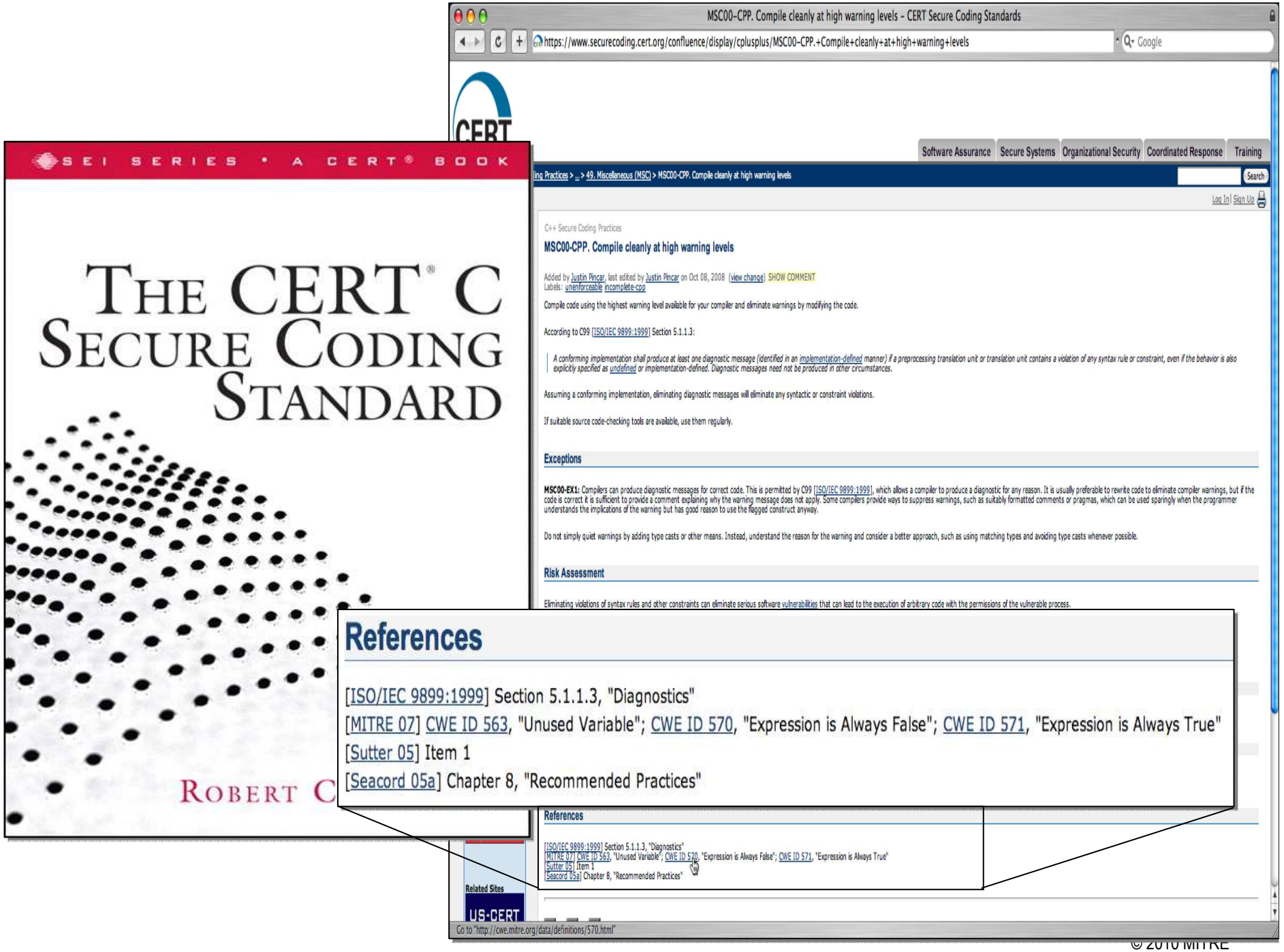
Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006

OWASP
The Open Web Application Security Project

OWASP Top 10 - 2010
The Ten Most Critical Web Application Security Risks

lease

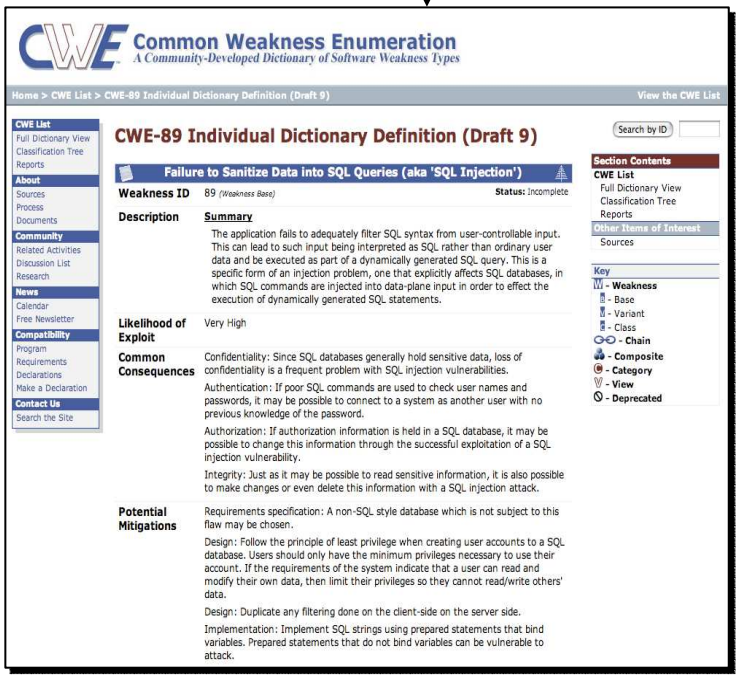



SEI SERIES • A CERT® BOOK
THE CERT® C SECURE CODING STANDARD
ROBERT C
MSC00-CPP. Compile cleanly at high warning levels - CERT Secure Coding Standards
https://www.securecoding.cert.org/confluence/display/cplusplus/MSC00-CPP.+Compile+cleanly+at+high+warning+levels
Google
CERT
Software Assurance
Secure Systems
Organizational Security
Coordinated Response
Training
ing Practices > .. > 49. Miscellaneous (MSC) > MSC00-CPP. Compile cleanly at high warning levels
Search
Log In | Sign Up
C++ Secure Coding Practices
MSC00-CPP. Compile cleanly at high warning levels
Added by Justin Pincar, last edited by Justin Pincar on Oct 08, 2008 (view change) SHOW COMMENT
Labels: unenforceable incomplete-cpp
Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.
According to C99 [ISO/IEC 9899:1999] Section 5.1.1.3:
A conforming implementation shall produce at least one diagnostic message (identified in an implementation-defined manner) if a preprocessing translation unit or translation unit contains a violation of any syntax rule or constraint, even if the behavior is also explicitly specified as undefined or implementation-defined. Diagnostic messages need not be produced in other circumstances.
Assuming a conforming implementation, eliminating diagnostic messages will eliminate any syntactic or constraint violations.
If suitable source code-checking tools are available, use them regularly.
Exceptions
MSC00-EX1: Compilers can produce diagnostic messages for correct code. This is permitted by C99 [ISO/IEC 9899:1999], which allows a compiler to produce a diagnostic for any reason. It is usually preferable to rewrite code to eliminate compiler warnings, but if the code is correct it is sufficient to provide a comment explaining why the warning message does not apply. Some compilers provide ways to suppress warnings, such as suitably formatted comments or pragmas, which can be used sparingly when the programmer understands the implications of the warning but has good reason to use the flagged construct anyway.
Do not simply quiet warnings by adding type casts or other means. Instead, understand the reason for the warning and consider a better approach, such as using matching types and avoiding type casts whenever possible.
Risk Assessment
Eliminating violations of syntax rules and other constraints can eliminate serious software vulnerabilities that can lead to the execution of arbitrary code with the permissions of the vulnerable process.
References
[ISO/IEC 9899:1999] Section 5.1.1.3, "Diagnostics"
[MITRE 07] CWE ID 563, "Unused Variable"; CWE ID 570, "Expression is Always False"; CWE ID 571, "Expression is Always True"
[Sutter 05] Item 1
[Seacord 05a] Chapter 8, "Recommended Practices"
Related Sites
US-CERT
Go to "http://cwe.mitre.org/data/definitions/570.html"
References
[ISO/IEC 9899:1999] Section 5.1.1.3, "Diagnostics"
[MITRE 07] CWE ID 563, "Unused Variable"; CWE ID 570, "Expression is Always False"; CWE ID 571, "Expression is Always True"
[Sutter 05] Item 1
[Seacord 05a] Chapter 8, "Recommended Practices"

Some High-Level CWEs Are Now Part of the NVD CVE Information

NVD XML feeds also include CWE

Vulnerability Type (View All)
SQL Injection (CWE-89)



Industry Uptake

Manually review code after security education

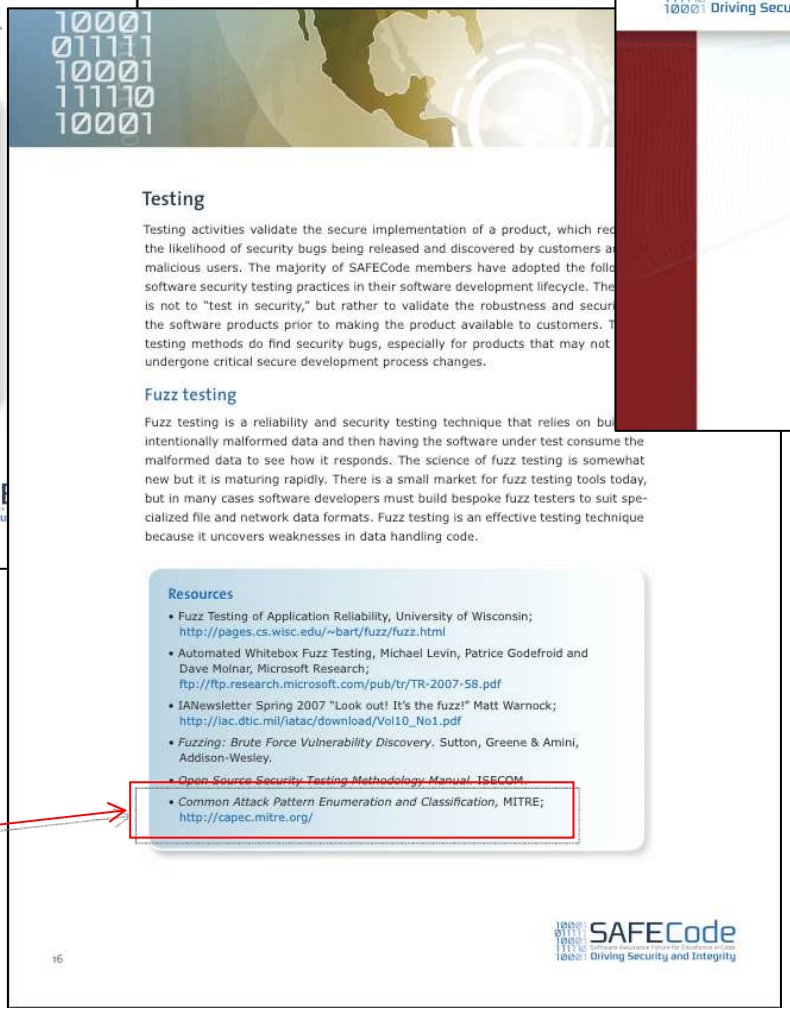
Manual code review, especially review of high-risk code, such as code that faces the Internet or parses data from the Internet, is critical, but only if the people performing the code review know what to look for and how to fix any code vulnerabilities they find. The best way to help understand classes of security bugs and remedies is education, which should minimally include the following areas:

- C and C++ vulnerabilities and remedies, most notably buffer overruns and integer arithmetic issues.
- Web-specific vulnerabilities and remedies, such as cross-site scripting (XSS).
- Database-specific vulnerabilities and remedies, such as SQL injection.
- Common cryptographic errors and remedies.

Many vulnerabilities are programming language (C, C++ etc) or domain-specific (web, database) and others can be categorized by vulnerability type, such as injection (XSS and SQL Injection) or cryptographic (poor random number generation and weak secret storage) so specific training in these areas is advised.

Resources

- A Process for Performing Security Code Reviews, Michael Howard, IEEE Security & Privacy July/August 2006.
- .NET Framework Security — Code Review; http://msdn.microsoft.com/en-us/library/aa302437.aspx
- *Common Weakness Enumeration*, MITRE; http://cwe.mitre.org/
- Security Code Reviews; http://www.codesecurely.org/Wiki/view.aspx/Security_Code_Reviews
- Security Code Review — Use Visual Studio Bookmarks To Capture Security Findings; http://blogs.msdn.com/alikl/archive/2008/01/24/security-code-review-use-visual-studio-bookmarks-to-capture-security-findings.aspx
- Security Code Review Guidelines, Adam Shostack; http://www.verber.com/mark/cs/security/code-review.html
- OSWASP Top Ten; http://www.owasp.org/index.php/OWASP_Top_Ten_Project



Testing

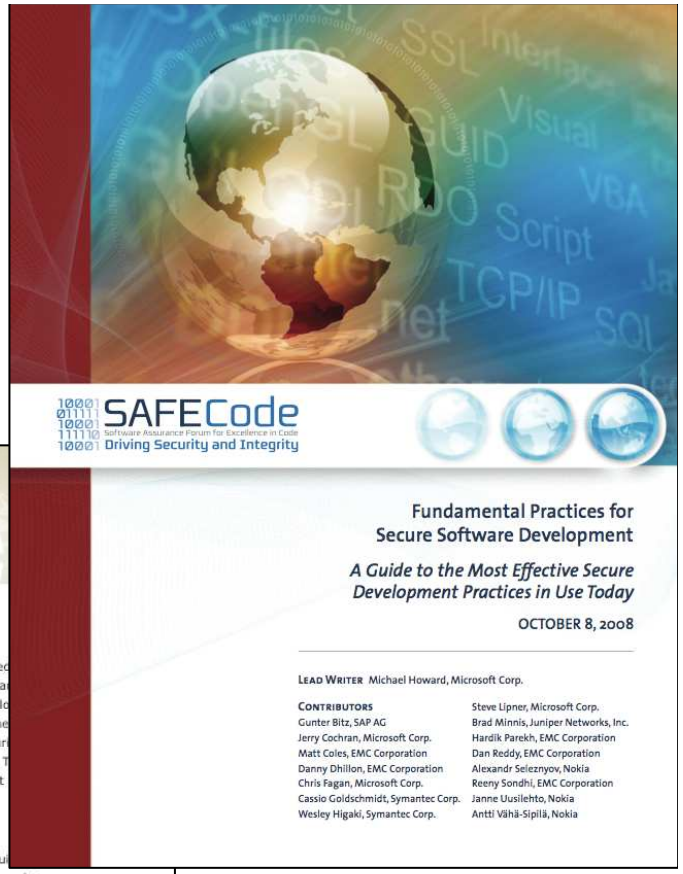
Testing activities validate the secure implementation of a product, which red
the likelihood of security bugs being released and discovered by customers a
malicious users. The majority of SAFECode members have adopted the follo
software security testing practices in their software development lifecycle. The
is not to "test in security," but rather to validate the robustness and securi
the software products prior to making the product available to customers. T
testing methods do find security bugs, especially for products that may not
undergone critical secure development process changes.

Fuzz testing

Fuzz testing is a reliability and security testing technique that relies on bu
intentionally malformed data and then having the software under test consume the
malformed data to see how it responds. The science of fuzz testing is somewhat
new but it is maturing rapidly. There is a small market for fuzz testing tools today,
but in many cases software developers must build bespoke fuzz testers to suit specialized file and network data formats. Fuzz testing is an effective testing technique
because it uncovers weaknesses in data handling code.

Resources

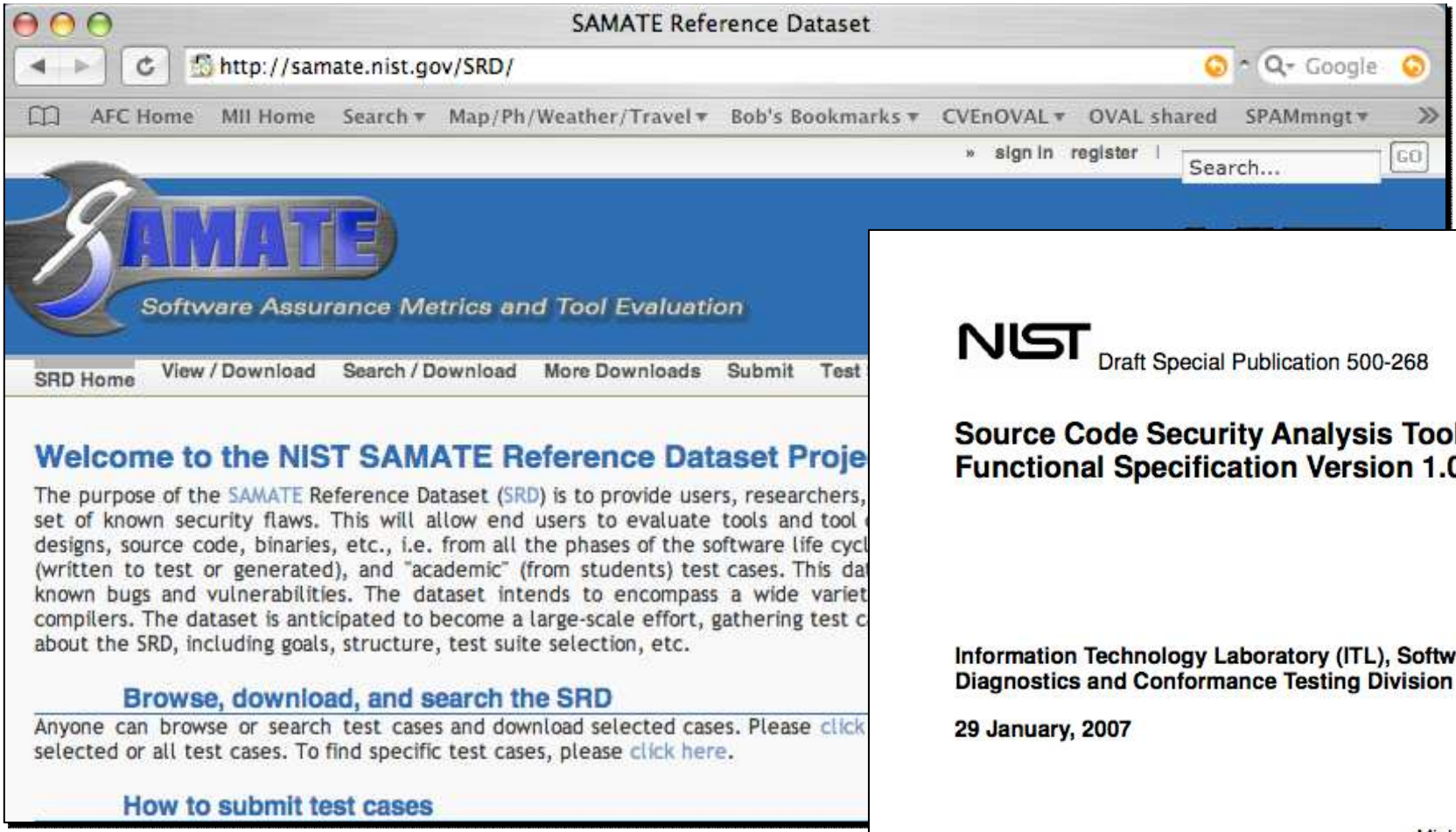
- Fuzz Testing of Application Reliability, University of Wisconsin; http://pages.cs.wisc.edu/~bart/fuzz/fuzz.html
- Automated Whitebox Fuzz Testing, Michael Levin, Patrice Godefroid and Dave Molnar, Microsoft Research; ftp://ftp.research.microsoft.com/pub/tr/TR-2007-58.pdf
- IANewsletter Spring 2007 "Look out! It's the fuzz!" Matt Warnock; http://iac.dtic.mil/iatac/download/Vol10_No1.pdf
- *Fuzzing: Brute Force Vulnerability Discovery*. Sutton, Greene & Amini, Addison-Wesley.
- *Open Source Security Testing Methodology Manual*. ISECOM.
- *Common Attack Pattern Enumeration and Classification*, MITRE; http://capec.mitre.org/

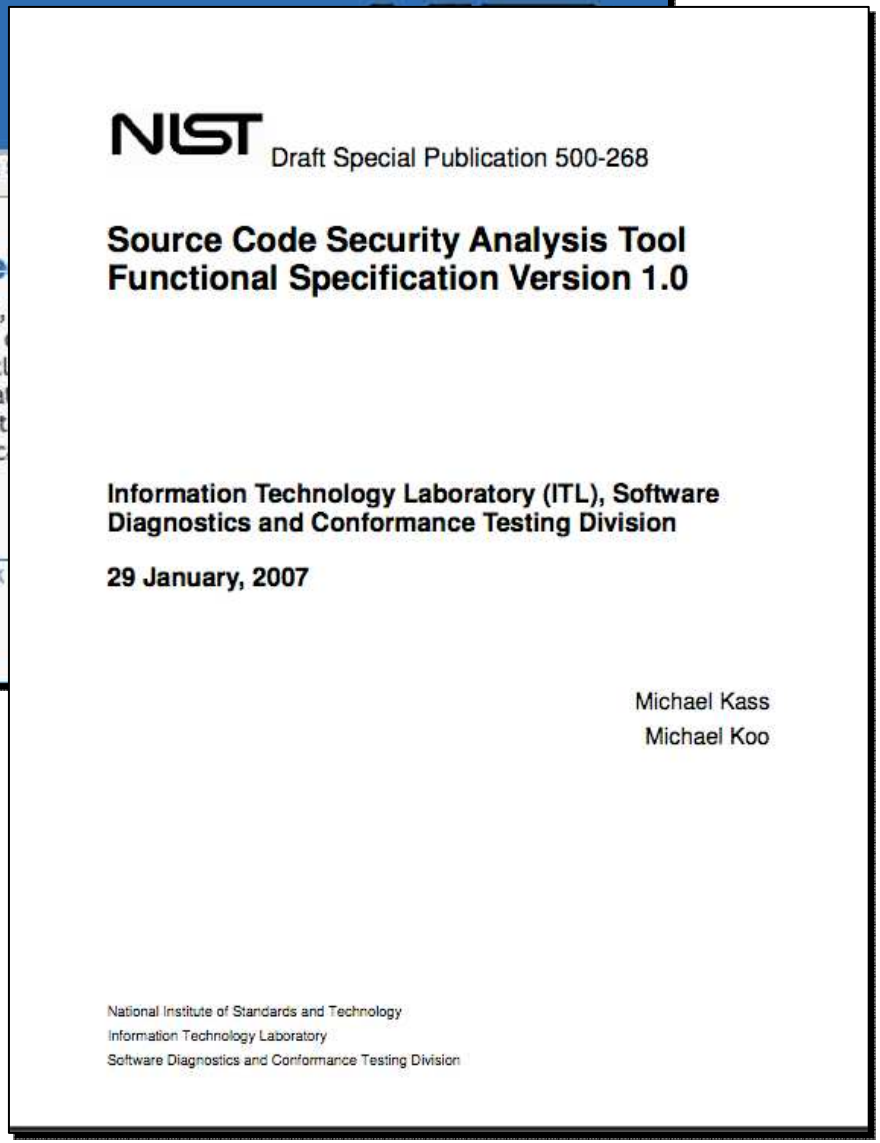


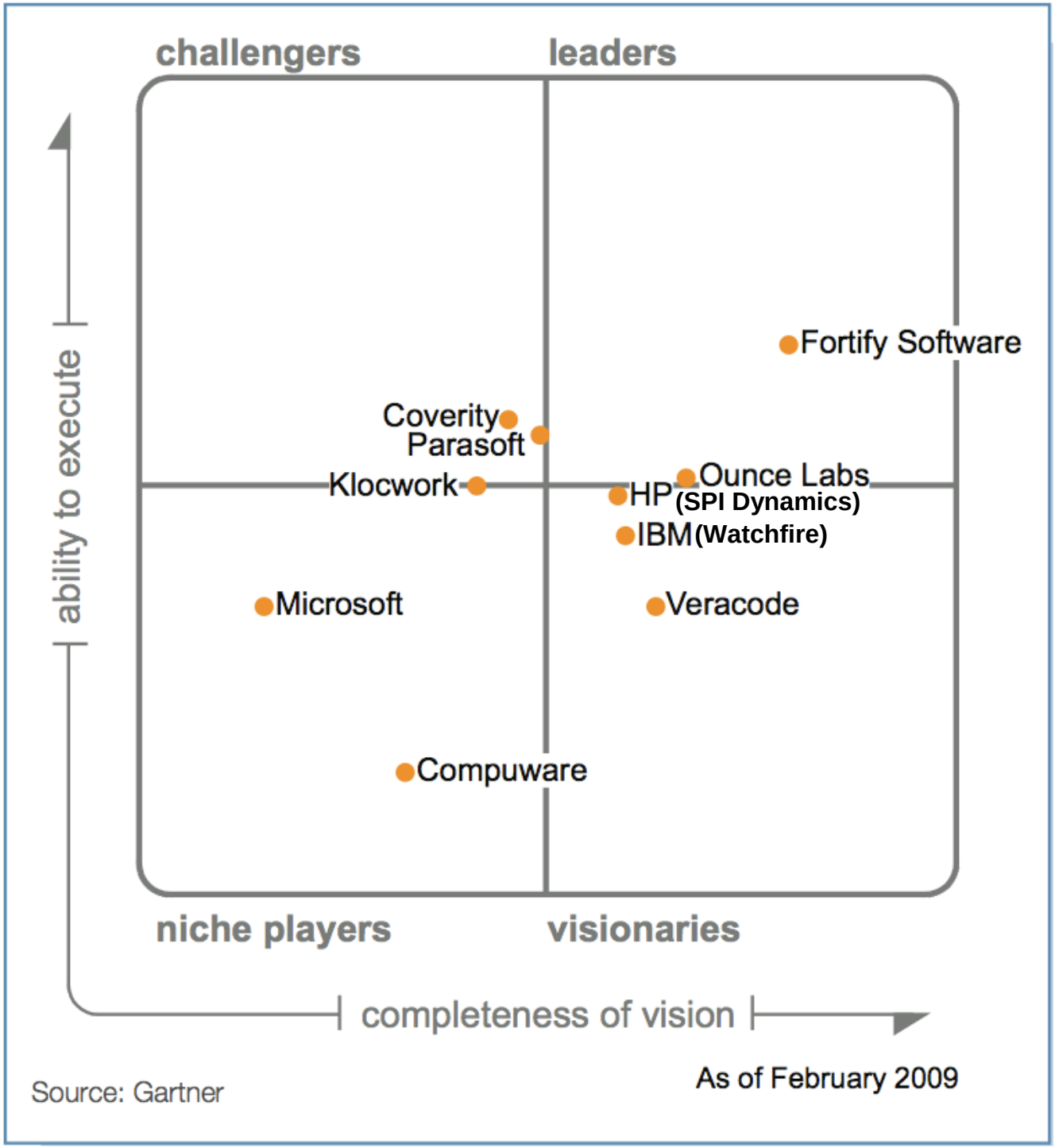
CWE

CAPEC


The Security Development Lifecycle : MS08-078 and the SDL
http://blogs.msdn.com/sdl/archive/2008/12/18/ms08-078-and-the-sdl.aspx
RSS
Google
Welcome to MSDN Blogs Sign in | Join | Help
Search
The Security Development Lifecycle
HOME
EMAIL
RSS 2.0
ATOM 1.0
Recent Posts
MS08-078 and the SDL
Announcing CAT.NET CTP and AntiXSS v3 beta
SDL videos
BlueHat SDL Sessions Wrap-up
Secure Coding Secrets?
Tags
Common Criteria Crawl Walk Run
Privacy SDL SDL Pro Network
Security Assurance Security Blackhat
SDL threat modeling
News
Blogroll
BlueHat Security Briefings
The Microsoft Security Response Center
Michael Howard's Web Log
The Data Privacy Imperative
Security Vulnerability Research & Defense
Visual Studio Code Analysis Blog
MSRC Ecosystem Strategy Team
Books / Papers / Guidance
The Security Development Lifecycle (Howard and Lipner)
Privacy Guidelines for Developing Software Products and Services
Microsoft Security Development Lifecycle (SDL) — Portal
Microsoft Security Development Lifecycle (SDL) — Process Guidance (Web)
Microsoft Security Development Lifecycle (SDL) — Process Guidance (.doc)
MS08-078 and the SDL
Hi, Michael here.
Every bug is an opportunity to learn, and the security update that fixed the data binding bug that affected Internet Explorer users is no exception.
The Common Vulnerabilities and Exposures (CVE) entry for this bug is CVE-2008-4844.
Before I get started, I want to explain the goals of the SDL and the security work here at Microsoft. The SDL is designed as a multi-layered process to help systemically reduce security vulnerabilities; if one component of the SDL process fails to prevent or catch a bug, then some other component should prevent or catch the bug. The SDL also mandates the use of security defenses whose impact will be reflected in the "mitigations" section of a security bulletin, because we know that no software development process will catch all security bugs. As we have said many times, the goal of the SDL is to "Reduce vulnerabilities, and reduce the severity of what's missed."
In this post, I want to focus on the SDL-required code analysis, code review, fuzzing and compiler and operating system defenses and how they fared.
Background
The bug was an invalid pointer dereference in MSHTML.DLL when the code handles data binding. It's important to point out that there is no heap corruption and there is no heap-based buffer overrun!
When data binding is used, IE creates an object which contains an array of data binding objects. In the code in question, when a data binding object is released, the array length is not correctly updated leading to a function call into freed memory.
The vulnerable code looks a little like this (by the way, the real array name is _aryPXfer, but I figured ArrayOfObjectsFromIE is a little more descriptive for people not in the Internet Explorer team.)
int MaxIdx = ArrayOfObjectsFromIE.Size()-1;
for (int i=0; i <= MaxIdx; i++) {
if (!ArrayOfObjectsFromIE[i])
continue;
ArrayOfObjectsFromIE[i]->TransferFromSource();
...
}
Here's how the vulnerability manifests itself: if there are two data transfers with the same identifier (so MaxIdx is 2), and the first transfer updates the length of the ArrayOfObjectsFromIE array when its work was done and releases its data binding object, the loop count would still be whatever MaxIdx was at the start of the loop, 2.
This is a time-of-check-time-of-use (TOCTOU) bug that led to code calling into a freed memory block. The Common Weakness Enumeration (CWE) classification for this vulnerability is CWE-367.
The fix was to check the maximum iteration count on each loop iteration rather than once before the loop
a time-of-check-time-of-use (TOCTOU) bug that led to code calling into a freed memory block. The
on Weakness Enumeration (CWE) classification for this vulnerability is CWE-367.
TOCTOU issues. We will update our training to address this.
Our static analysis tools don't find this because the tools would need to understand the re-entrant nature of the code.
Fuzz Testing
September 2008 (5)
August 2008 (2)
July 2008 (8)
June 2008 (4)


SAMATE Reference Dataset
http://samate.nist.gov/SRD/
Google
AFC Home
MII Home
Search
Map/Ph/Weather/Travel
Bob's Bookmarks
CVEnOVAL
OVAL shared
SPAMmngt
sign in
register
Search...
GO
SAMATE
Software Assurance Metrics and Tool Evaluation
SRD Home
View / Download
Search / Download
More Downloads
Submit
Test
Welcome to the NIST SAMATE Reference Dataset Proje
The purpose of the SAMATE Reference Dataset (SRD) is to provide users, researchers,
set of known security flaws. This will allow end users to evaluate tools and tool
designs, source code, binaries, etc., i.e. from all the phases of the software life cycl
(written to test or generated), and "academic" (from students) test cases. This da
known bugs and vulnerabilities. The dataset intends to encompass a wide variet
compilers. The dataset is anticipated to become a large-scale effort, gathering test c
about the SRD, including goals, structure, test suite selection, etc.
Browse, download, and search the SRD
Anyone can browse or search test cases and download selected cases. Please click
selected or all test cases. To find specific test cases, please click here.
How to submit test cases


NIST
Draft Special Publication 500-268
Source Code Security Analysis Tool
Functional Specification Version 1.0
Information Technology Laboratory (ITL), Software
Diagnostics and Conformance Testing Division
29 January, 2007
Michael Kass
Michael Koo
National Institute of Standards and Technology
Information Technology Laboratory
Software Diagnostics and Conformance Testing Division



Gartner Magic Quadrant for Static Application Security Testing Tools

Plus Some Other Important Tool Players…

Cenzic
CAST Software
Polyspace
Security Innovation
LDRA
KDM Analytics
SureLogic
Programming Research Inc
Armorize
SofCheck
GrammaTech

CWE Compatibility & Effectiveness Program

(launched Feb 2007)

Organizations Participating

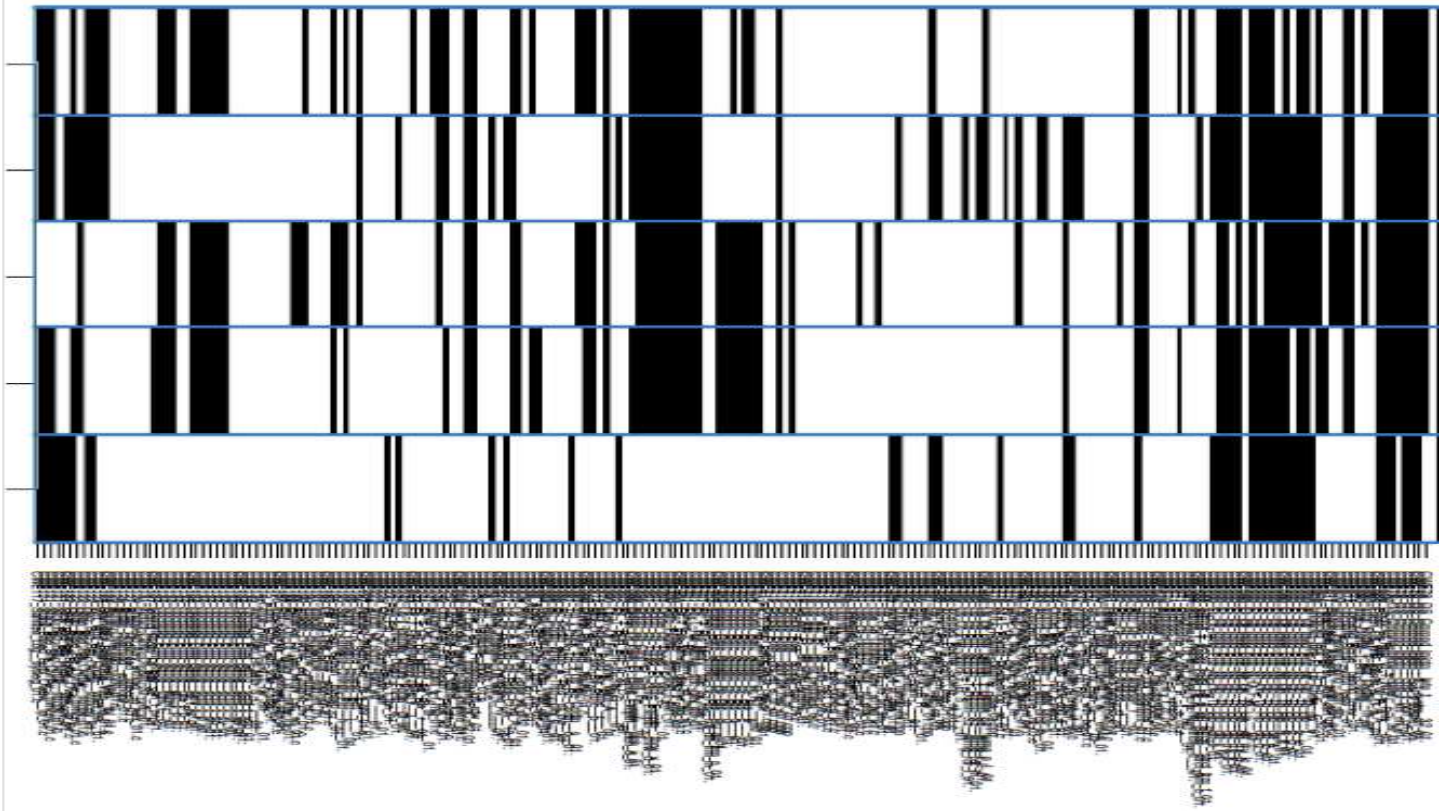
cwe.mitre.org/compatible/

All organizations participating in the CWE Compatibility and Effectiveness Program are listed below, including those with CWE-Compatible Products and Services and those with Declarations to Be CWE-Compatible.

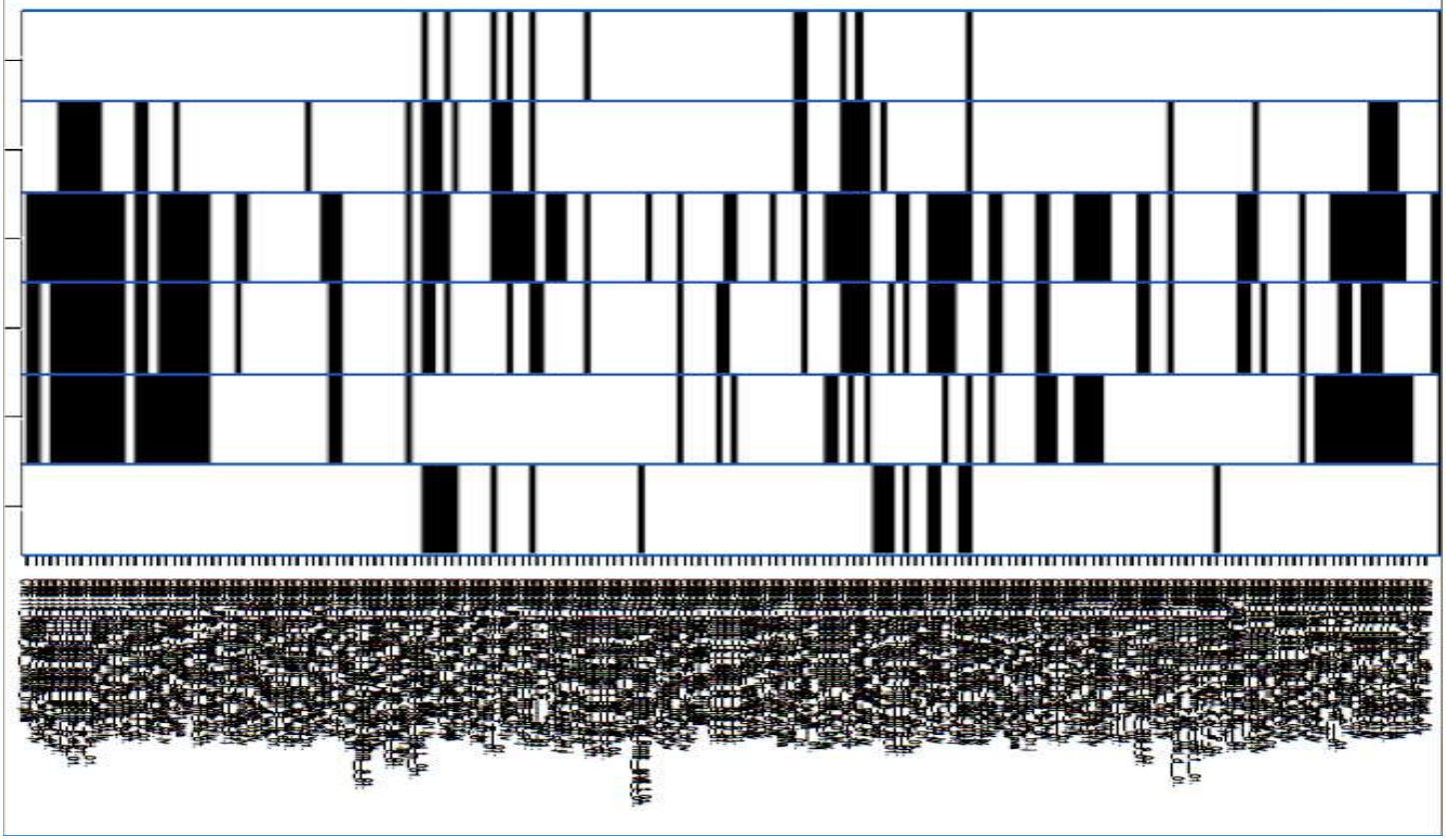
TOTALS
Organizations Participating: **28**
Products & Services: **47**

Products are listed alphabetically by organization name:

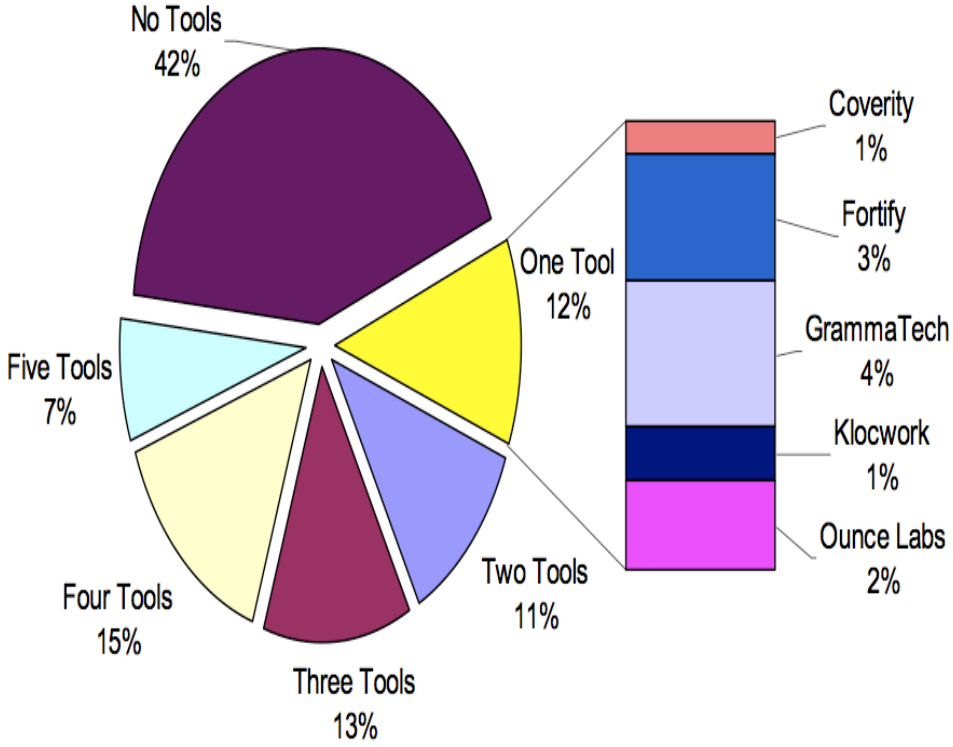
C Test Cases



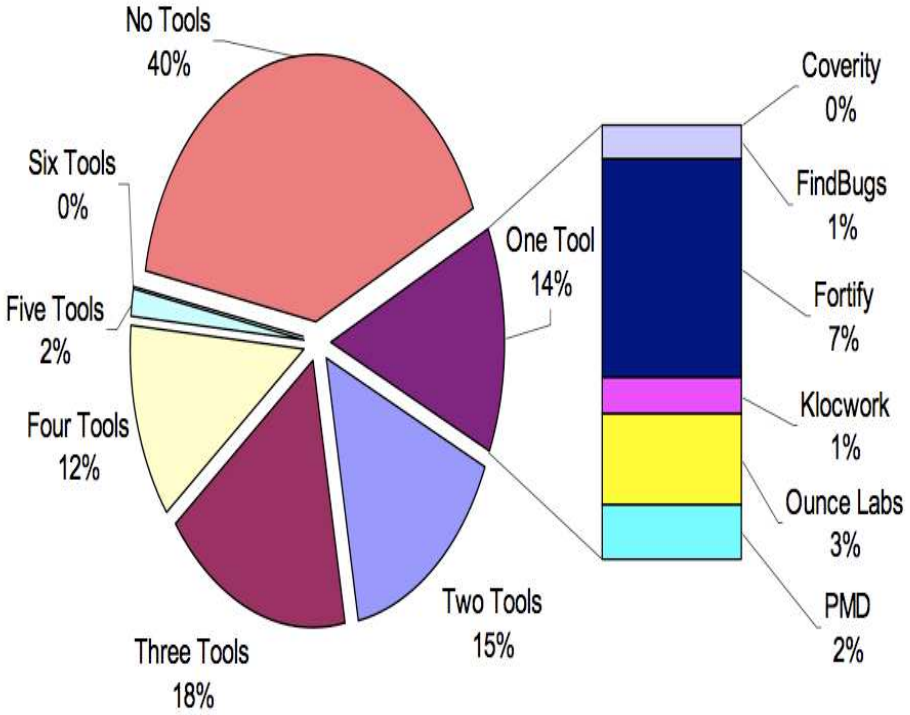
Java Test Cases



C/C++ "Breadth" Test Case Coverage



Java "Breadth" Test Case Coverage



2 March 2009 Jointly funded by:
Assistant Secretary of the Navy Chief System Engineer
197 Isaac Hull
Washington Navy Yard, DC
and Naval Ordnance Safety & Security Activity
Box 47, Bldg D-323
3817 Strauss Avenue Indian Head, MD 20640
Prepared by:Booz Allen Hamilton McLean, VA

Software Security Tools

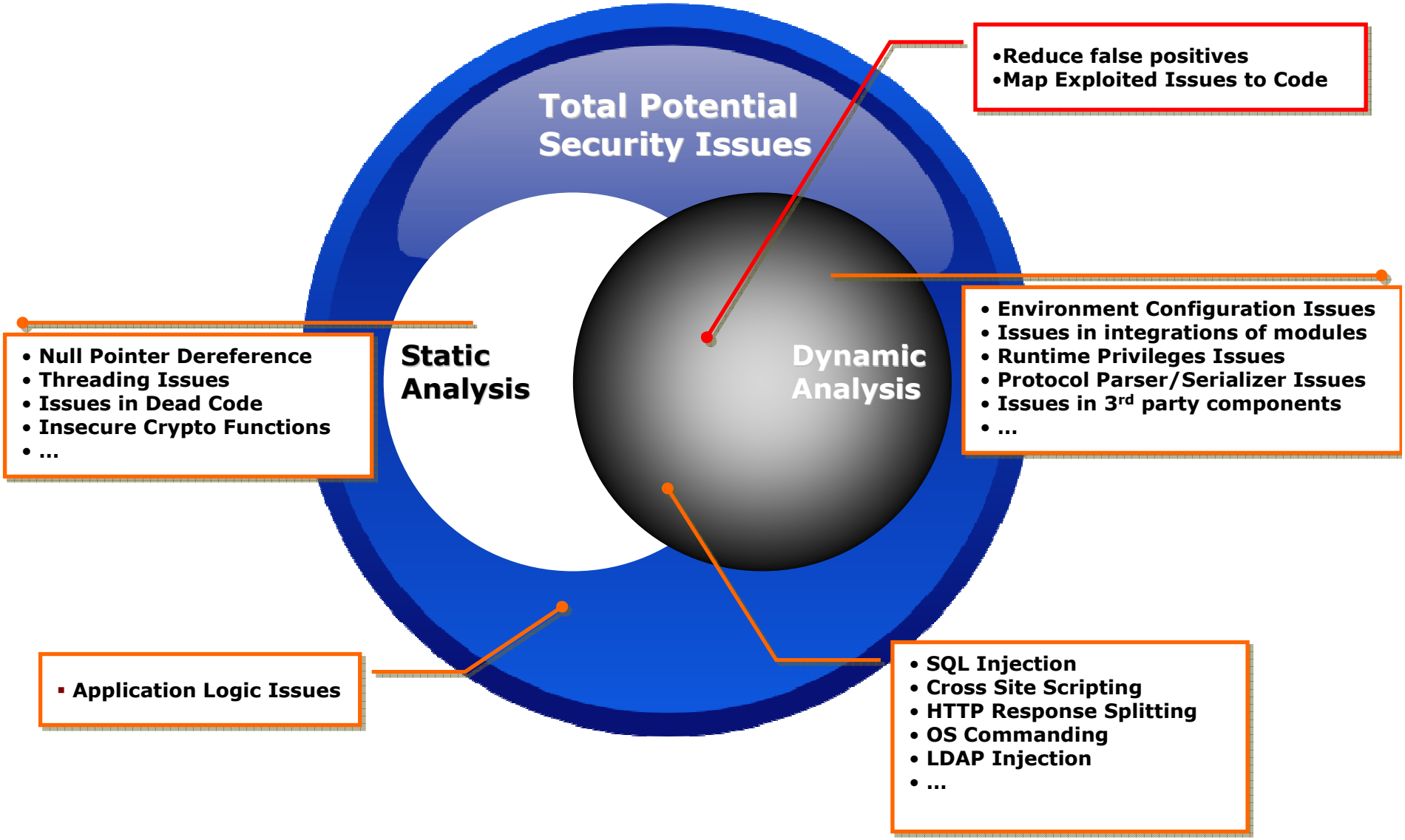
- Static Analysis
- Source Code Fault Injection
- Dynamic Analysis
- Architectural Analysis
- Pedigree Analysis
- Binary Code Analysis
- Disassembler Analysis
- Binary Fault Injection
- Fuzzing
- Malicious Code Detectors
- Bytecode Analysis

Software Security Assessment Tools Review

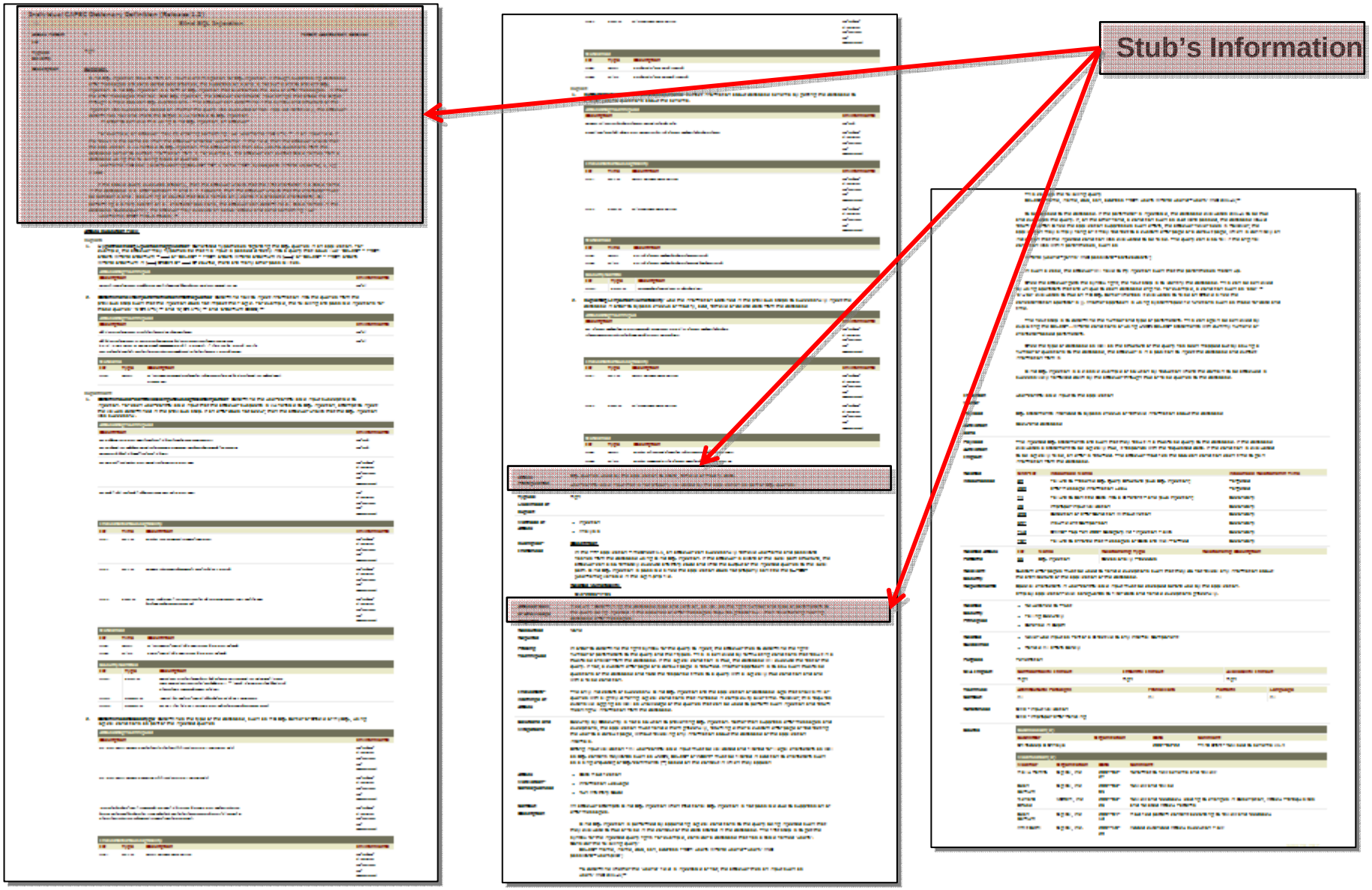
Summary of Evaluation **Key:** X*- To be most beneficial X+ - In some cases X% - When possible X# - e.g., compilation	Static Analysis Code Scanning	Source Code Fault Injection	Dynamic Analysis	Architectural Analysis	Pedigree Analysis	Binary Code Analysis	Disassembler Analysis	Binary Fault Injection	Fuzzing	Malicious Code Detector	Byte Code Analysis
When to Use											
Requirements				X							
Design				X*							
Implementation	X*	X	X	X	X*	X	X	X	X	X	X*
Testing	X	X*	X*	X	X	X	X	X*	X*	X	X
Production	X	X	X	X	X	X	X	X	X	X	X
Acquisition	X	X	X	X	X	X*	X*	X	X	X*	X
Required Skills (Understanding of…)											
Underlying source code		X									
Underlying development methodology			X#	X					X		
Implementing language	X	X					X%				
Binary						X+	X				
Bytecode											X
Testing methodology		X	X	X			X		X	X	X
Benefits											
Reduces cost over system life	X	X	X	X	X	X	X	X	X	X	X
Educates developers about secure programming	X			X	X						
Rechecks legacy code	X				X	X	X	X	X		X
Automates repetitive and tedious aspects of source code security audits	X				X						
Checks for good programming style	X										X
Increased test coverage		X						X	X		
Increased accuracy		X							X		
No need for source code			X	X		X	X	X	X	X	X
Improved accuracy and coverage		X	X					X	X		
Reduces the amount of testing necessary					X						
No disassembly	X	X	X	X	X	X		X	X		
Guaranteed the analysis is performed on the actual product			X+	X		X	X	X	X	X	X
Drawbacks											
No architectural-level flaws	X		X		X	X	X			X	X
Thorough understanding of the software			X	X				X	X		
Required expertise		X		X			X	X	X		
Requires use of open source software					X						
Lack of tool availability		X				X				X	
Licensing concerns							X				
Reliance on a primary vendor							X				
Additional analysis		X		X			X	X	X		X
Additional preparation		X	X	X					X		
Limited to a single language											X

Table 1:Evaluation Summary of Analyses for Source Code, Executables, and Intermediate Representations

Value of Aligning Multiple Perspectives



Complete CAPEC Entry Information



Linkage with Fundamental Changes in Enterprise Security Initiatives

Twenty Critical Controls for Effective Cyber Def Guidelines

What the 20 CSC Critics say...

20 Critical Security Controls - Version 2.0

- 20 Critical Security Controls - Introduction (Version 2.0)
- Critical Control 1: Inventory of Authorized and Unauthorized
- Critical Control 2: Inventory of Authorized and Unauthorized
- Critical Control 3: Secure Configurations for Hardware and So Servers
- Critical Control 4: Secure Configurations for Network Devices
- Critical Control 5: Boundary Defense
- Critical Control 6: Maintenance, Monitoring, and Analysis of
- Critical Control 7: Application Software Security
- Critical Control 8: Controlled Use of Administrative Privilege
- Critical Control 9: Controlled Access Based on Need to Know
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr

CAG: Critical Control 7: Application Software Security

<< previous control | Consensus Audit Guidelines | next control >>

How do attackers exploit the lack of this control?

Attacks against vulnerabilities in web-based and other application software have been a top priority for criminal organizations in recent years. Application software that does not properly check the size of user input, fails to sanitize user input by filtering out unneeded but potentially malicious character sequences, or does not initialize and clear variables properly could be vulnerable to remote compromise. Attackers can inject specific exploits, including buffer overflows, SQL injection attacks, and cross-site scripting code to gain control over vulnerable machines. In one attack in 2008, more than 1 million web servers were exploited and turned into infection engines for visitors to those sites using SQL injection. During that attack, trusted websites from state governments and other organizations compromised by attackers were used to infect hundreds of thousands of

CWE and CAPEC included in Control 7 of the "Twenty Critical Controls for Effective Cyber Defense: Consensus Audit Guidelines"

Procedures and tools for implementing t

Source code testing tools, web application security scanning tools, and object code testing tools have proven useful in securing application software, along with manual application security penetration testing by testers who have extensive programming knowledge as well as application penetration testing expertise. The Common Weakness Enumeration (CWE) initiative is utilized by many such tools to identify the weaknesses that they find. Organizations can also use CWE to determine which types of weaknesses they are most interested in addressing and removing. A broad community effort to identify the "Top 25 Most Dangerous Programming Errors" is also available as a minimum set of important issues to investigate and address during the application development process. When evaluating the effectiveness of testing for these weaknesses, the Common Attack Pattern Enumeration and Classification (CAPEC) can be used to organize and record the breadth of the testing for the CWEs as well as a way for testers to think like attackers in their development of test cases.

Correlate, Integrate, Automate

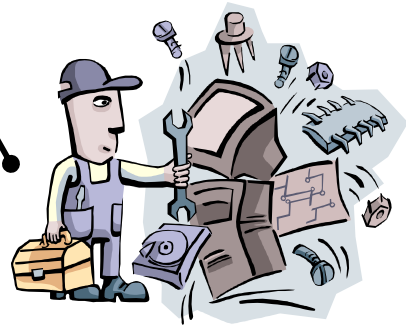
Threats

MAEC™

Detection

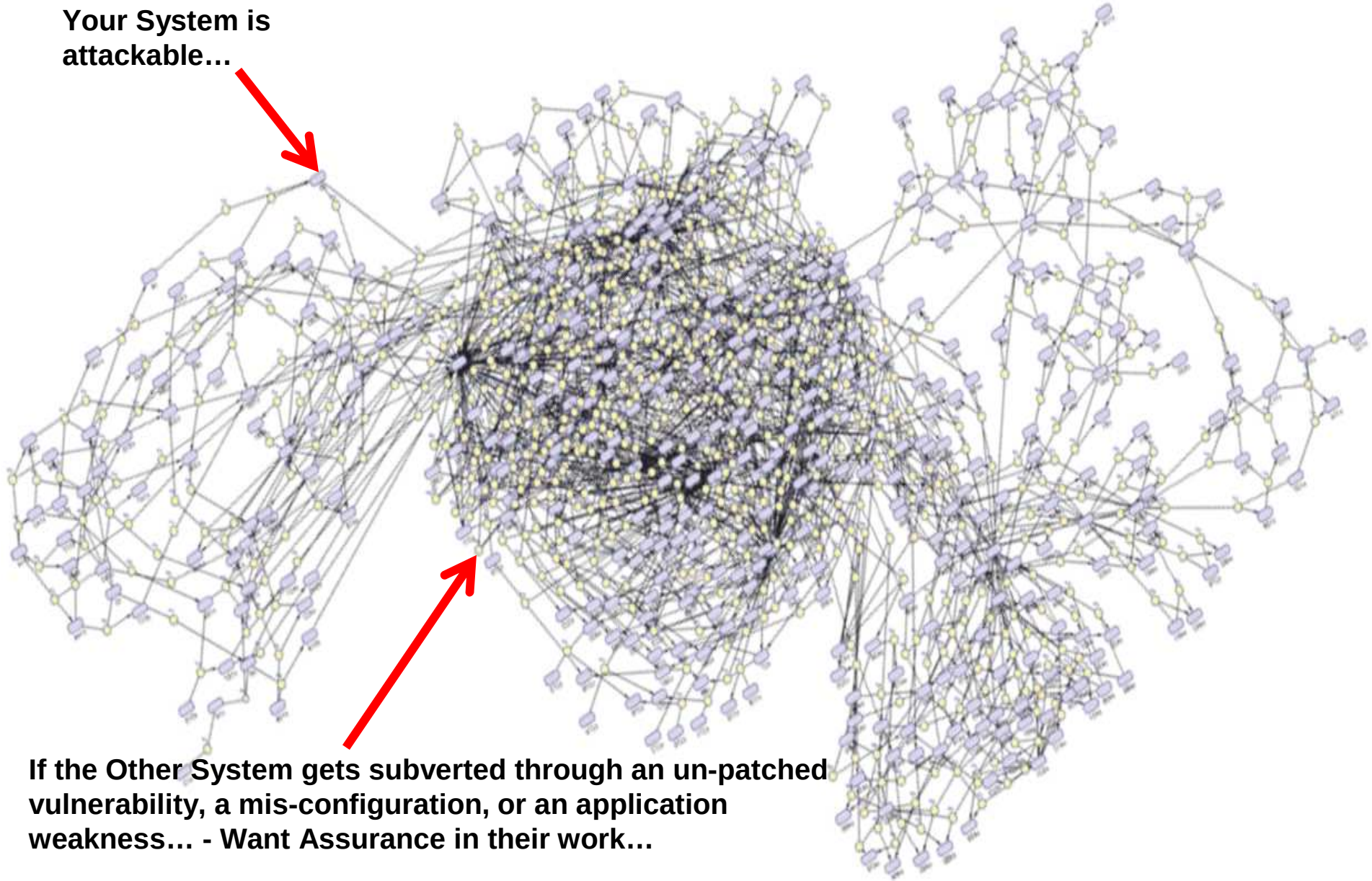
Vulnerabilities

Platforms

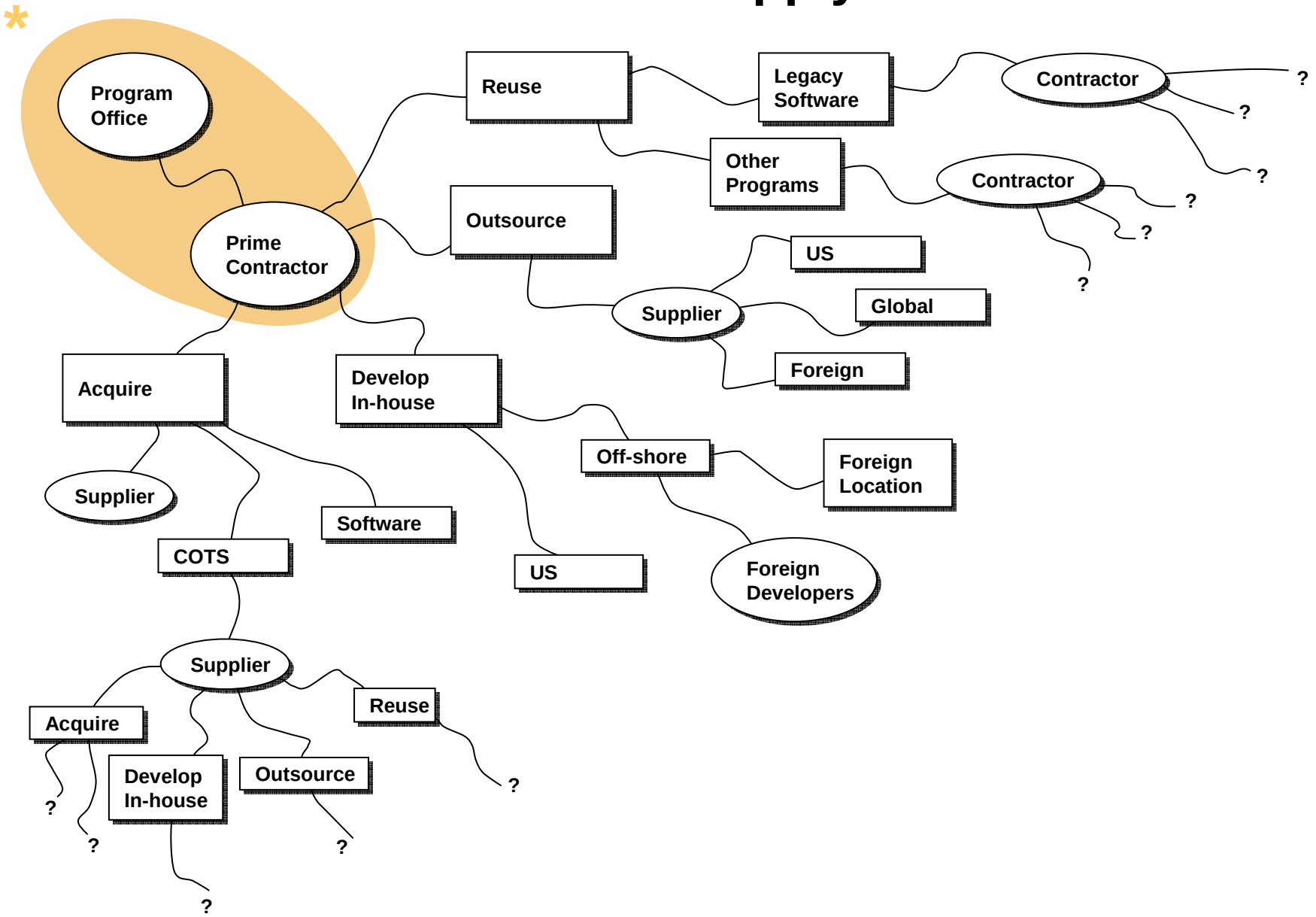


Response

Today Everything's Connected



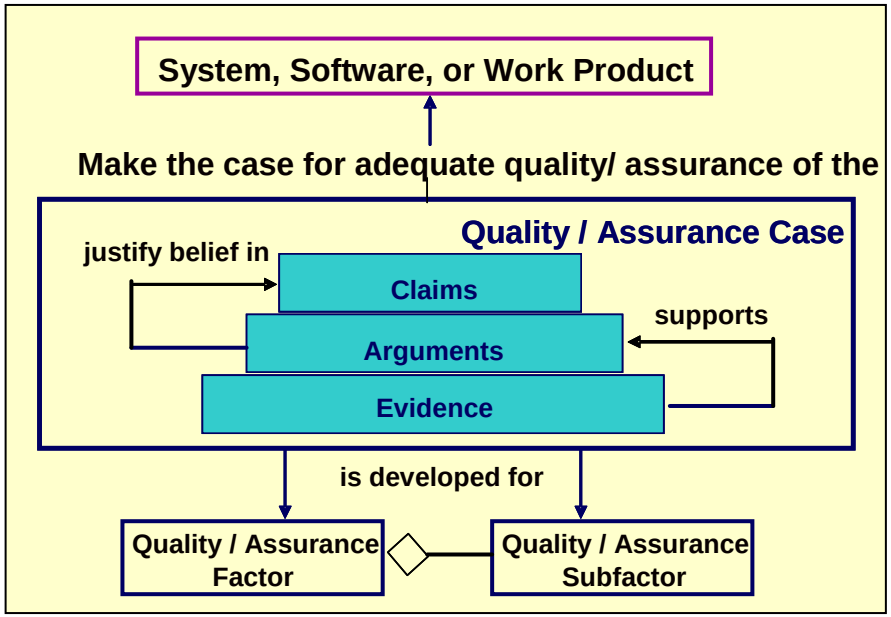
The Software Supply Chain



* "Scope of Supplier Expansion and Foreign Involvement" graphic in DACS www.softwaretechnews.com Secure Software Engineering, July 2005 article "Software Development Security: A Risk Management Perspective" synopsis of May 2004 GAO-04-678 report "Defense Acquisition: Knowledge of Software Suppliers Needed to Manage Risks"

ISO/IEC/IEEE 15026 Assurance Case

- **Set of structured assurance claims, supported by evidence and reasoning (arguments), that demonstrates how assurance needs have been satisfied.**
  - Shows compliance with assurance objectives
  - Provides an argument for the safety and security of the product or service.
  - Built, collected, and maintained throughout the life cycle
  - Derived from multiple sources



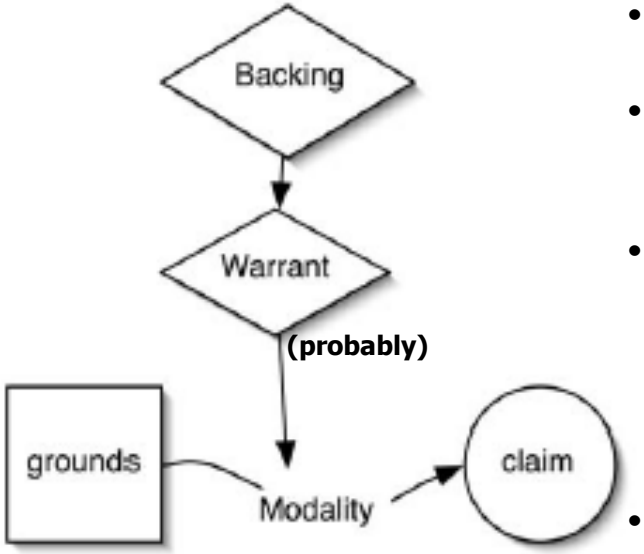
- **Sub-parts**
  - A high level summary
  - Justification that product or service is acceptably safe, secure, or dependable
  - Rationale for claiming a specified level of safety and security
  - Conformance with relevant standards & regulatory requirements
  - The configuration baseline
  - Identified hazards and threats and residual risk of each hazard / threat
  - Operational & support assumptions

Attributes

- Clear
- Consistent
- Complete
- Comprehensible
- Defensible
- Bounded
- Addresses all life cycle stages

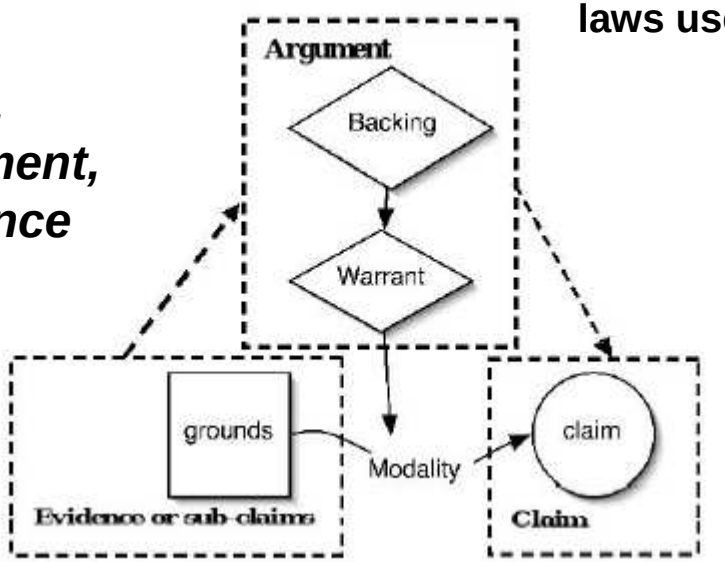
Assurance Claims with Support by 'Substantial' Reasoning

Stephen Toulmin, 1958

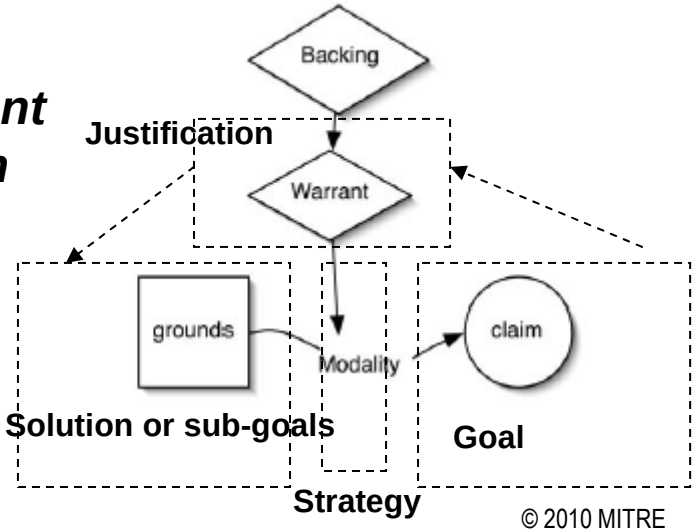


- **Claims are assertions put forward for general acceptance**
- **The justification for claim is based on some grounds, the "specific facts about a precise situation that clarify and make good for a claim"**
- **The basis of the reasoning from the grounds (the facts) to the claim is articulated. Toulmin coined the term "warrant" for "substantial argument". These are statements indicating the general ways of argument being applied in a particular case and implicitly relied on and whose trustworthiness is well established".**
- **The basis of the warrant might be questioned, so "backing" for the warrant may be introduced. Backing might be the validation of the scientific and engineering laws used**

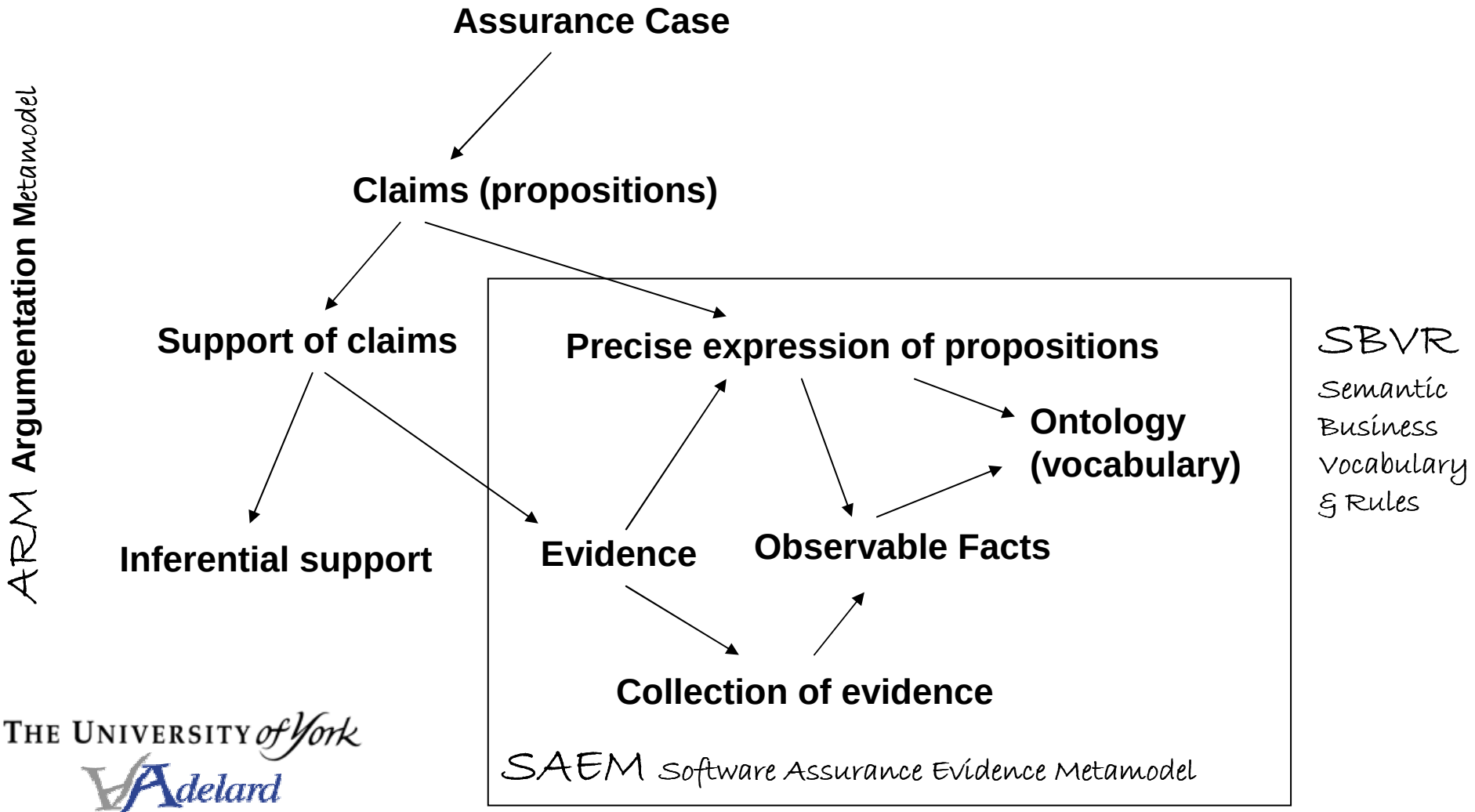
CAE
Claim,
Argument,
Evidence



GSN
Goal
Statement
Notation



Object Management Group (OMG)
Systems Assurance Task Force
Claims-Evidence-Arguments Overview



THE UNIVERSITY of York

Adelard

KDM Analytics

Software Assurance Ecosystem: The Formal Framework

The value of formalization extends beyond software systems to include related software system process, people and documentation

Process Docs & Artifacts

Requirements/Design Docs & Artifacts

Reports
Risk Analysis, etc)

Process, People & Documentation Evaluation Environment

- Some point tools to assist evaluators but mainly manual work
- Claims in Formal SBVR vocabulary
- Evidence in Formal SBVR vocabulary
- Large scope requires large effort

Process, People, documentation Evidence

Formalized Specifications

Claims, Arguments and Evidence Repository

- Formalized in SBVR vocabulary
- Automated verification of claims against evidence
- Highly automated and sophisticated risk assessments using transitive inter-evidence point relationships

Software System / Architecture Evaluation

- Many integrated & highly automated tools to assist evaluators
- Claims and Evidence in Formal vocabulary
- Combination of tools and ISO/OMG standards
- Standardized SW System Representation In KDM
- Large scope capable (system of systems)
- Iterative extraction and analysis for rules

Software system Technical Evidence

Executable Specifications

Software System Artifacts

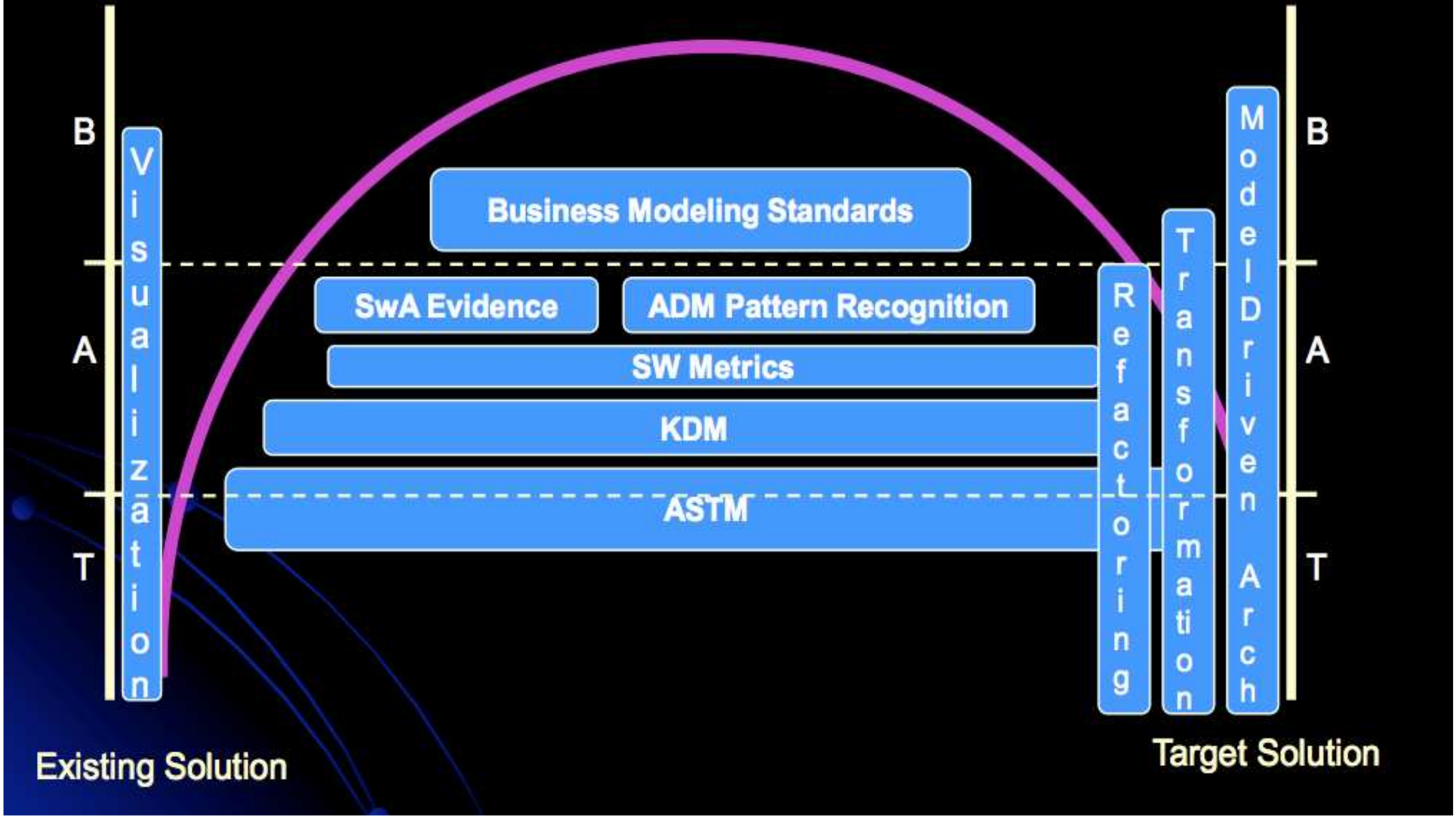
Hardware Environment

Protection Profiles

IA Controls

CWE

ADM Standards Span the Entire IT Architecture Spectrum



SC27 WG3

Common Criteria v4 CCDB
- **TOE to leverage CAPEC & CWE**
- **Also investigating how to leverage ISO/IEC 15026**

NIAP Evaluation Scheme
- **Above plus**
- **Also investigating how to leverage SCAP**

ISO IEC

ISO/IEC JTC 1/SC 27 **Nxxxx**

ISO/IEC JTC 1/SC 27/WG x **Nxxxxx**

REPLACES: N

ISO/IEC JTC 1/SC 27

Information technology - Security techniques

Secretariat: DIN, Germany

DOC TYPE: NB NWI Proposal for a technical report (TR)

TITLE: **National Body New Work Item Proposal on "Secure software development and evaluation under ISO/IEC 15408 and ISO/IEC 18405"**

SOURCE: INCITS/C31, National Body of (US)

DATE: 2009-09-30

PROJECT: **15408 and 18405**

STATUS: This document is circulated for consideration at the forthcoming meeting of SC 27/WG 3 to be held in Redmond (WA, USA) on 2nd – 6th November 2009.

ACTION ID: ACT

DUE DATE:

DISTRIBUTION: P-, O- and L-Members
W. Fumy, SC 27 Chairman
M. De Soete, SC 27 Vice-Chair
E. J. Humphreys, K. Naemura, M. Bañón, M.-C. Kang, K. Rannenberg, WG-Conveners

MEDIUM: Livelink-server

NO. OF PAGES: xx

Secretariat ISO/IEC JTC 1/SC 27 –
DIN Deutsches Institut für Normung e. V., Burggrafenstr. 6, 10772 Berlin, Germany
Telephone: + 49 30 2601-2652; Facsimile: + 49 30 2601-42723; E-mail: krystyna.passia@din.de
HTTP://www.jtc1sc27.din.de/en

New Work Item Proposal

NP submitting

PROPOSAL FOR A NEW WORK ITEM

Date of presentation of proposal: YYYY-MM-DD	Proposer: ISO/IEC JTC 1 SC27
Secretariat: National Body	ISO/IEC JTC 1 N XXXX ISO/IEC JTC 1/SC 27 N

A proposal for a new work item shall be submitted to the secretariat of the ISO/IEC joint technical committee concerned with a copy to the ISO Central Secretariat.

Presentation of the proposal

Title Secure software development and evaluation under ISO/IEC 15408 and ISO/IEC 18405

Scope

In the case where a target of evaluation (TOE) being evaluated under ISO/IEC 15408 and ISO/IEC 18405, includes specific software portions, the TOE developer may optionally present the developer's technical rationale for mitigating software common attack patterns and related weaknesses as described in the latest revision of the Common Attack Pattern Enumeration and Classification (CAPEC) available from http://capec.mitre.org/. The developer's technical rationale is expected to include a range of mitigation techniques, from architectural properties to design features, coding techniques, use of tools or other means.

This Technical Report (TR) provides guidance for the developer and the evaluator on how to use the CAPEC as a technical reference point during the TOE development life cycle and in an evaluation of the TOE secure software under ISO/IEC 15408 and 18405, by addressing:

a) A refinement of the IS 15408 Attack Potential calculation table for software, taking into account the entries contained in the CAPEC and their characterization.

b) How the information for mitigating software common attack patterns and related weaknesses is used in an IS 15408 evaluation, in particular providing guidance on how to determine which attack patterns and weaknesses are applicable to the TOE, taking into consideration of

1. the TOE technology;
2. the TOE security problem definition;
3. the interfaces the TOE exports that can be used by potential attackers;
4. the Attack Potentials that the TOE needs to provide resistance for.

c) How the technical rationale provided by the developer for mitigating software common attack patterns and related weaknesses is used in the evaluation of the TOE design and the development of test cases.

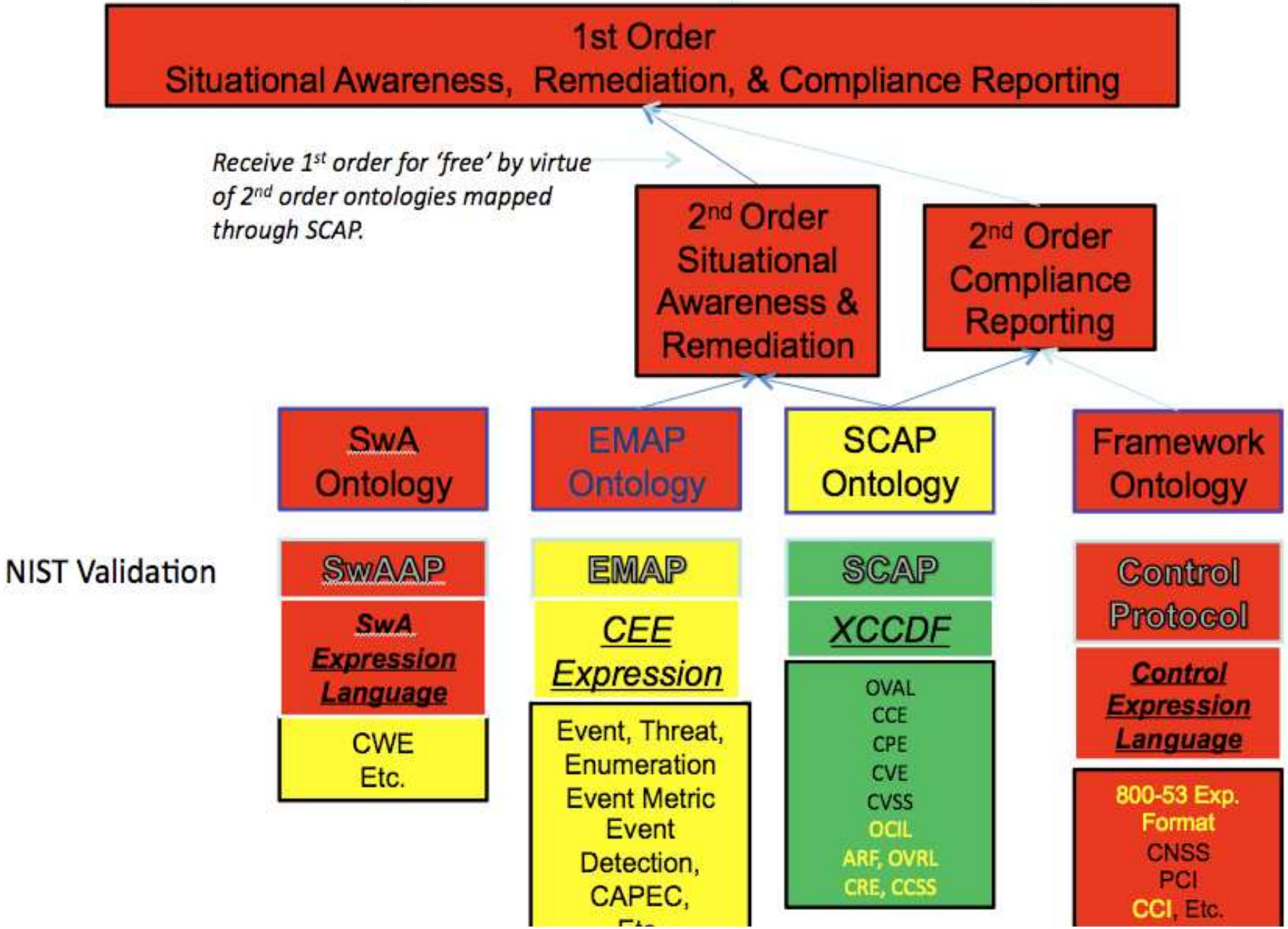
d) How the CAPEC and related Common Weakness Enumeration (CWE) taxonomies are used by the evaluator, who needs to consider all the applicable attack patterns and be able to exploit specific related software weaknesses while performing the subsequent vulnerability analysis (AVA_VAN) activities on the TOE.

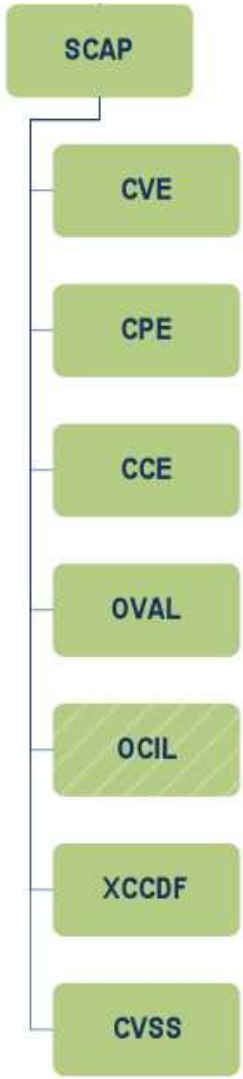
e) How incomplete entries from the CAPEC are resolved during an IS 15408 evaluation.

f) How the evaluator's attack and weakness analysis of the TOE incorporates other attacks and weaknesses not yet documented in the CAPEC.

The TR also investigates specific elements from the ISO/IEC 15026 (and its revision) are applicable to the guidelines being developed in the TR within the context of IS 15408 and 18405.

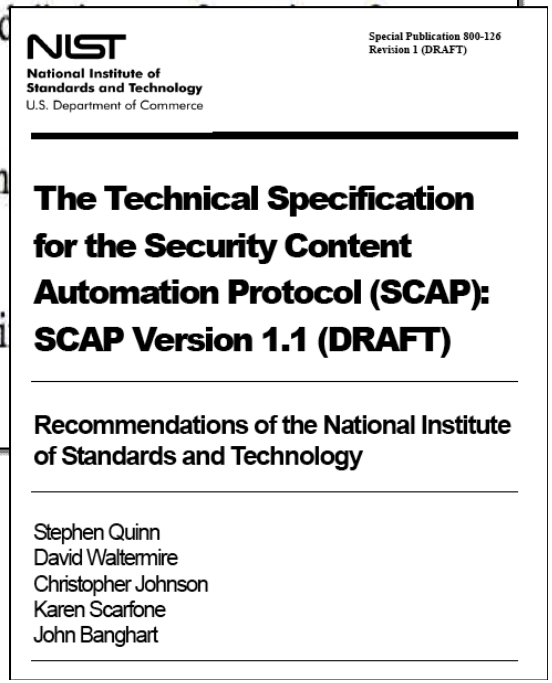
Security Automation (Current State)





SCAP 1.1 uses the following specifications:

- Extensible Configuration Checklist Description Format (XCCDF) 1.1.4, a language for authoring security checklists/benchmarks and for reporting results of checklist evaluation [QUI08]
- Open Vulnerability and Assessment Language (OVAL) 5.6, a language for representing system configuration information, assessing machine state, and reporting assessment results
- Open Checklist Interactive Language (OCIL) 2.0, a language for representing security checks that requires human feedback
- Common Platform Enumeration (CPE) 2.2, a nomenclature and dictionary of hardware, operating systems, and applications [BUT09]
- Common Configuration Enumeration (CCE) 5, a nomenclature and configurations
- Common Vulnerabilities and Exposures (CVE), a nomenclature an software flaws[9]
- Common Vulnerability Scoring System (CVSS) 2.0, an open speci severity of software flaw vulnerabilities [MEL07].



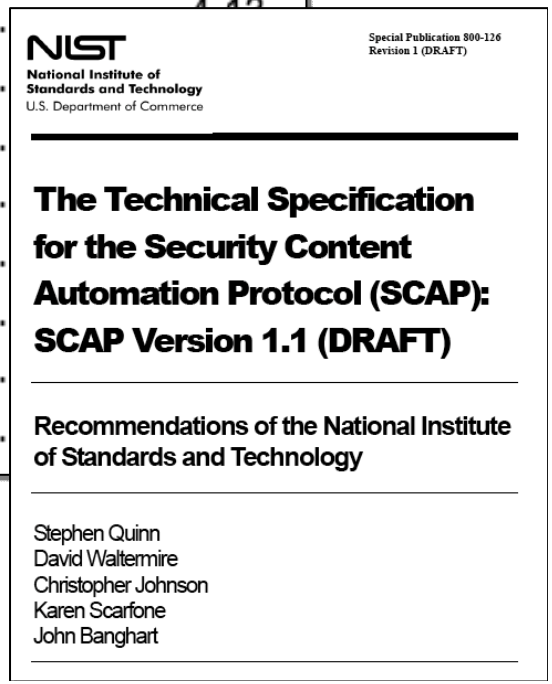
NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-126
Revision 1 (DRAFT)

The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.1 (DRAFT)

Recommendations of the National Institute of Standards and Technology

Stephen Quinn
David Waltermire
Christopher Johnson
Karen Scarfone
John Banghart



NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-126
Revision 1 (DRAFT)

The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.1 (DRAFT)

Recommendations of the National Institute of Standards and Technology

Stephen Quinn
David Waltermire
Christopher Johnson
Karen Scarfone
John Banghart

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-126
Revision 1 (DRAFT)

The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.1 (DRAFT)

Recommendations of the National Institute of Standards and Technology

Stephen Quinn
David Waltermire
Christopher Johnson
Karen Scarfone
John Banghart

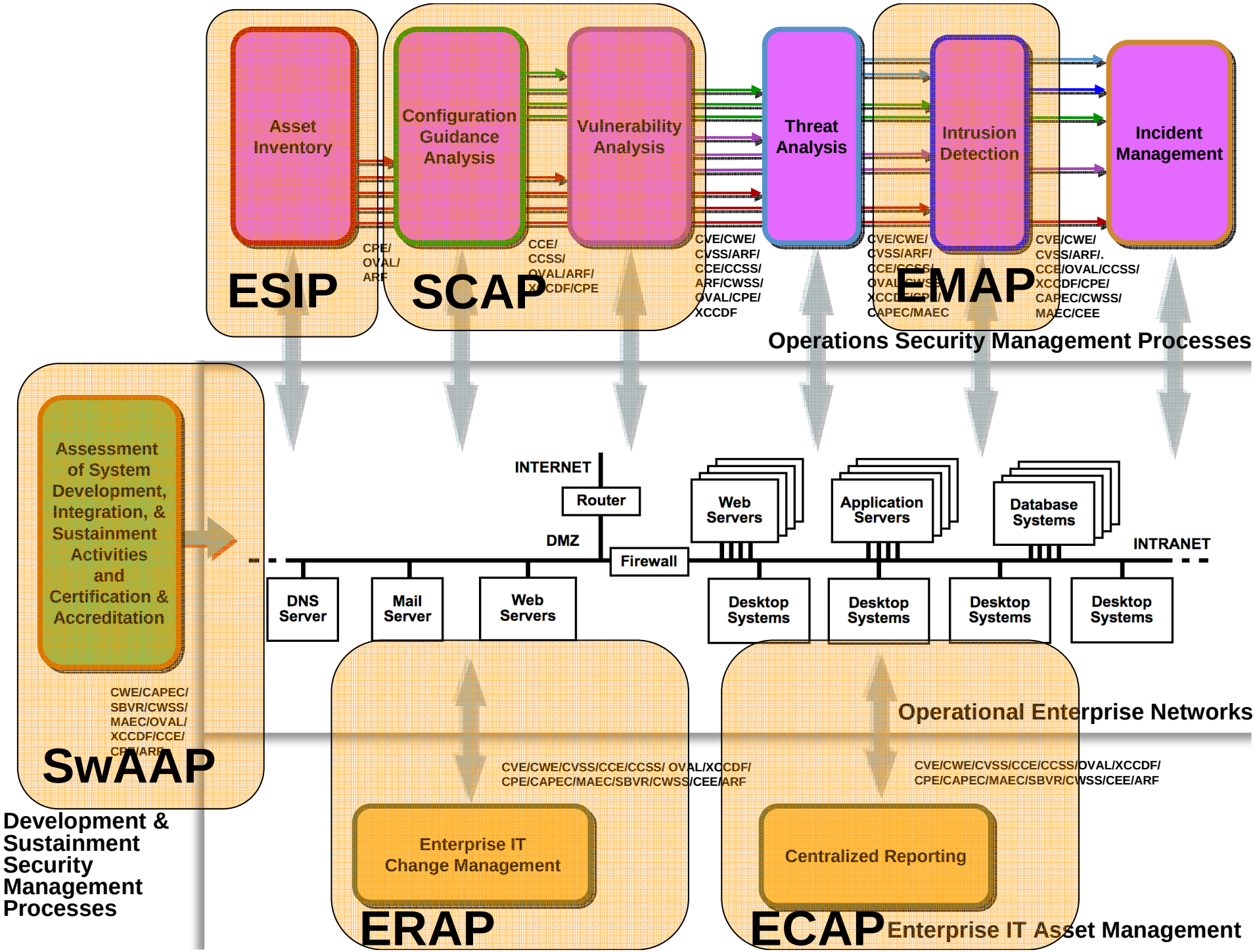
- **Software Assurance Automation Protocol (SwAAP)**
  - For measuring & enumerating software weaknesses and the assurance cases.

Common Weakness Enumeration (CWE),
Common Attack Pattern Enumeration & Classification (CAPEC),
Malware Attribute Enumeration & Characterization (MAEC),
Common Weakness Scoring System (CWSS),
OMG Software Assurance Evidence Metamodel (OMG SAEM),
OMG Arguementation Metatmodel (OMG ARG),
Software Assurance Findings Expression Schema (SAFES),
NIST SAMATE's "Food Label",
OMG Structured Metrics Metamodel (OMG SMM),
ISO "Assurance Case" 15026 (ISO 15026),
OMG Knowledge Discovery Metamodel (OMG KDM),
OMG Abstract Syntax Tree Metamodel (OMG ASTM)

- plus SCAP to capture "accredited" system CPEs and CCE settings?
- OVAL checks for capturing "finger print" of software applications to address supply-chain risk measurement?

"Other" Automation Protocols ("O"AP)

- Event Management Automation Protocol (EMAP)
  - **For reporting of security events. Common Event Expression (CEE), Malware Attribute Enumeration & Characterization (MAEC), and CAPEC.**
- Enterprise Remediation Automation Protocol (ERAP)
  - **For automated remediation of mis-configuration & missing patches. Common Remediation Enumeration (CRE) and Extended Remediation Information (ERI).**
- Enterprise Compliance Automation Protocol (ECAP)
  - **For reporting configuration compliance. Asset Reporting Format (ARF), Open Checklist Reporting Language (OCRL), etc.**
- Enterprise System Information Protocol (ESIP)
  - **For reporting of asset inventory information.**


Asset Inventory
Configuration Guidance Analysis
Vulnerability Analysis
Threat Analysis
Intrusion Detection
Incident Management
ESIP
SCAP
EMAP
CPE/ OVAL/ ARF
CCE/ CCSS/ OVAL/ARF/ XCCDF/CPE
CVE/CWE/ CVSS/ARF/ CCE/CCSS/ ARF/CWSS/ OVAL/CPE/ XCCDF
CVE/CWE/ CVSS/ARF/ CCE/CCSS/ OVAL/CWSS/ XCCDF/CPE/ CAPEC/MAEC
CVE/CWE/ CVSS/ARF/. CCE/OVAL/CCSS/ XCCDF/CPE/ CAPEC/CWSS/ MAEC/CEE
Operations Security Management Processes
Assessment of System Development, Integration, & Sustainment Activities and Certification & Accreditation
CWE/CAPEC/ SBVR/CWSS/ MAEC/OVAL/ XCCDF/CCE/ CPE/ARF
SwAAP
Development & Sustainment Security Management Processes
INTERNET
Router
DMZ
Firewall
Web Servers
Application Servers
Database Systems
INTRANET
DNS Server
Mail Server
Web Servers
Desktop Systems
Desktop Systems
Desktop Systems
Desktop Systems
Operational Enterprise Networks
CVE/CWE/CVSS/CCE/CCSS/ OVAL/XCCDF/ CPE/CAPEC/MAEC/SBVR/CWSS/CEE/ARF
CVE/CWE/CVSS/CCE/CCSS/OVAL/XCCDF/ CPE/CAPEC/MAEC/SBVR/CWSS/CEE/ARF
Enterprise IT Change Management
Centralized Reporting
ERAP
ECAP
Enterprise IT Asset Management


Questions?
ramartin@mitre.org