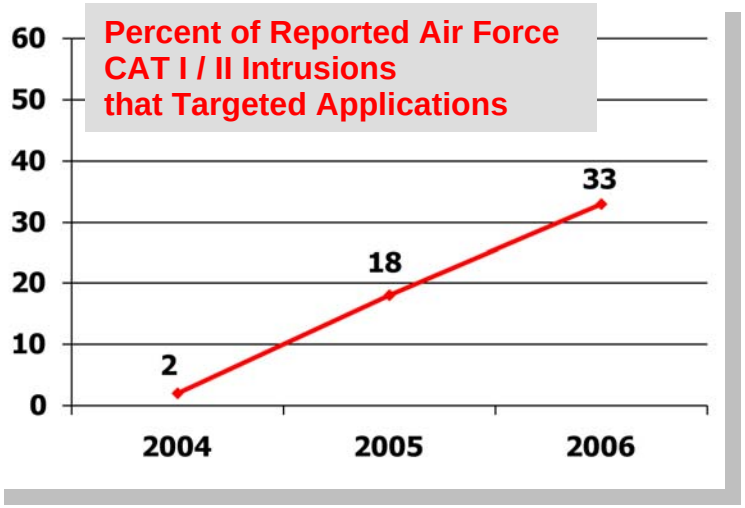
Software Assurance (SwA) Automation…

Robert A. Martin, MITRE
781-271-3001
ramartin@mitre.org

MITRE

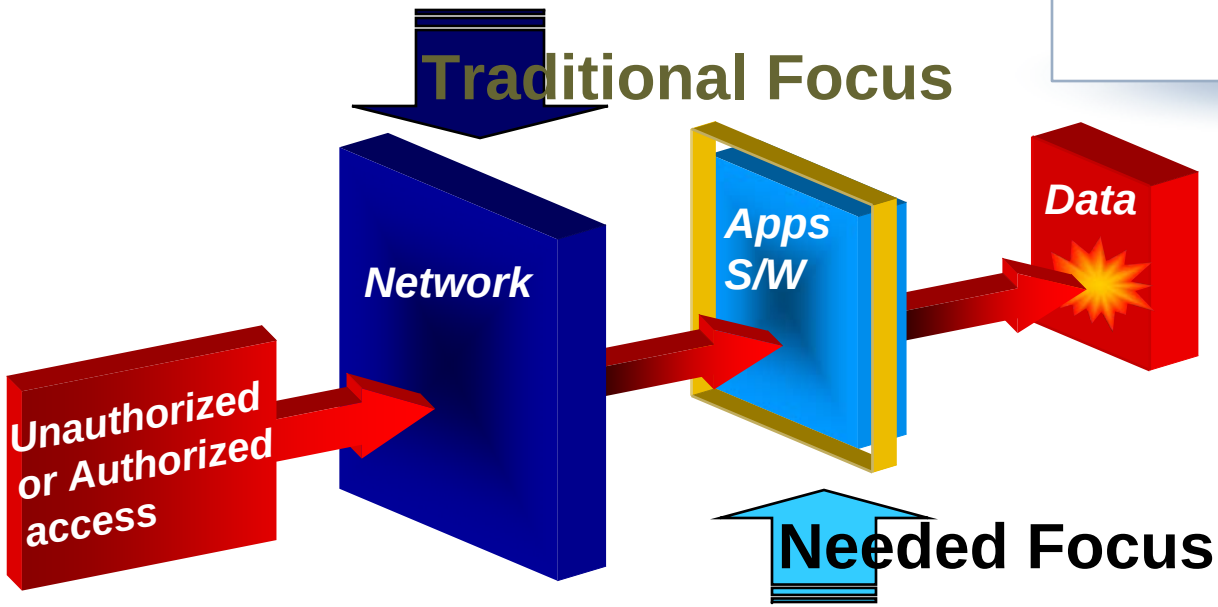
Making Security Measurable™

Application Security Wake-up Call



"Network-layer security mechanisms dominate current deployments but are proving inadequate in the face of more frequent application-layer attacks. This condition requires that vendors and users alike increase their focus on application-oriented security controls." - ***Mark Bouchard, META Group***

"75 % of hacks occur at the application level..."
Dec 2005
Gartner



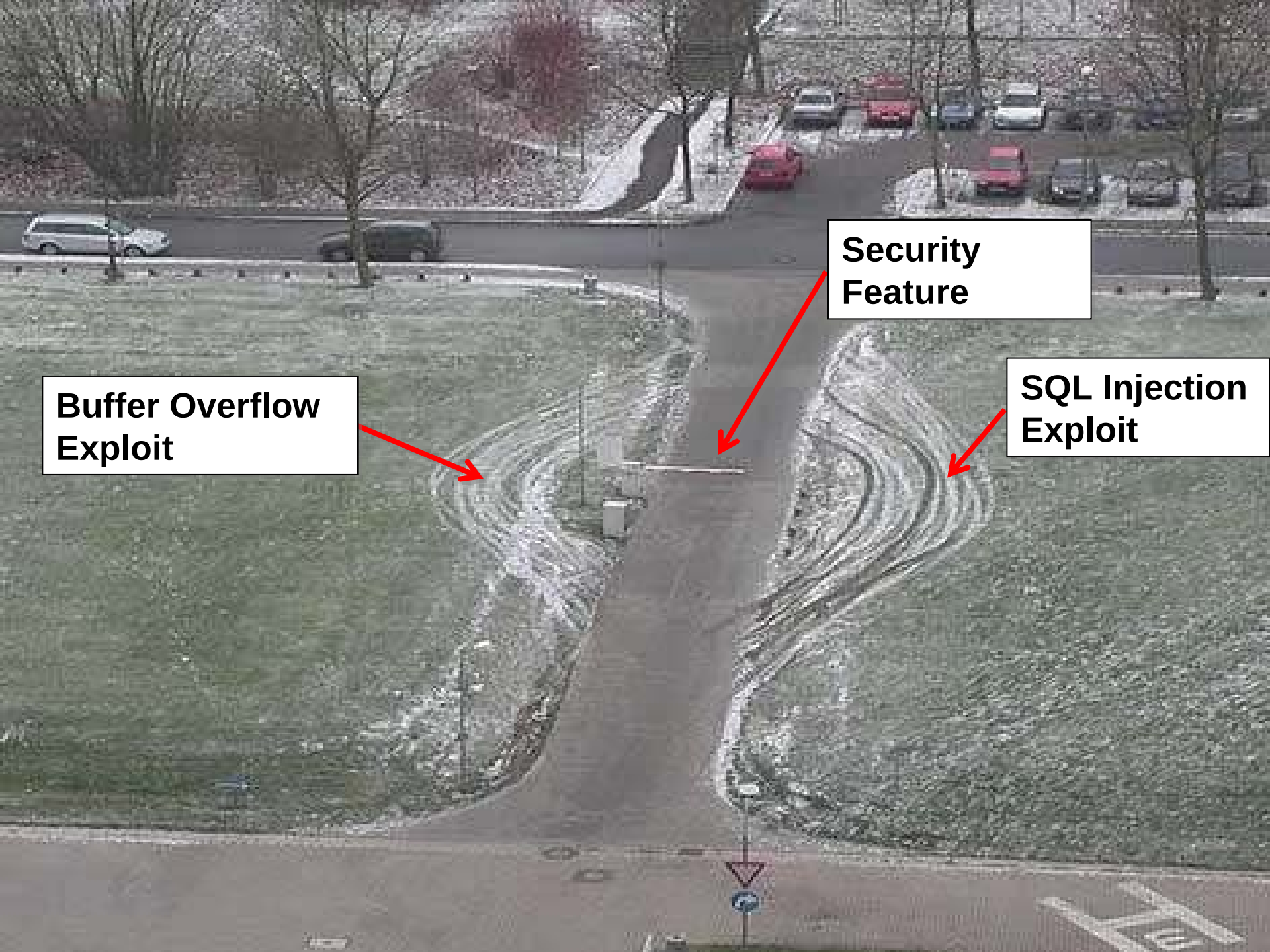
Catastrophic Failures Can Be Due To Software Weaknesses

```
... declare
      vertical_veloc_sensor: float;
      horizontal_veloc_sensor: float;
      vertical_veloc_bias: integer;
      horizontal_veloc_bias: integer;
 ... begin
declare
      pragma suppress(numeric_error,
      horizontal_veloc_bias);
begin sensor_get(vertical_veloc_sensor);
      sensor_get(horizontal_veloc_sensor);
      vertical_veloc_bias :=
      integer(vertical_veloc_sensor);
      horizontal_veloc_bias :=
      integer(horizontal_veloc_sensor);

 ... exception when numeric_error =>
      calculate_vertical_veloc();
      when others => use_irs1();
      end;
 end irs2;
```

- **A 64 floating point to 16 bit signed integer overflow condition?**
- **Poor exception handling?**
- **A faulty design assumption?**
- **Incomplete Testing process?**
- **A Software Reuse Error?**
- **Malicious Flaw Insertion?**

Software Flaws
Can Have Major Mission Impacts
- Ariane 5 Flight 501 -

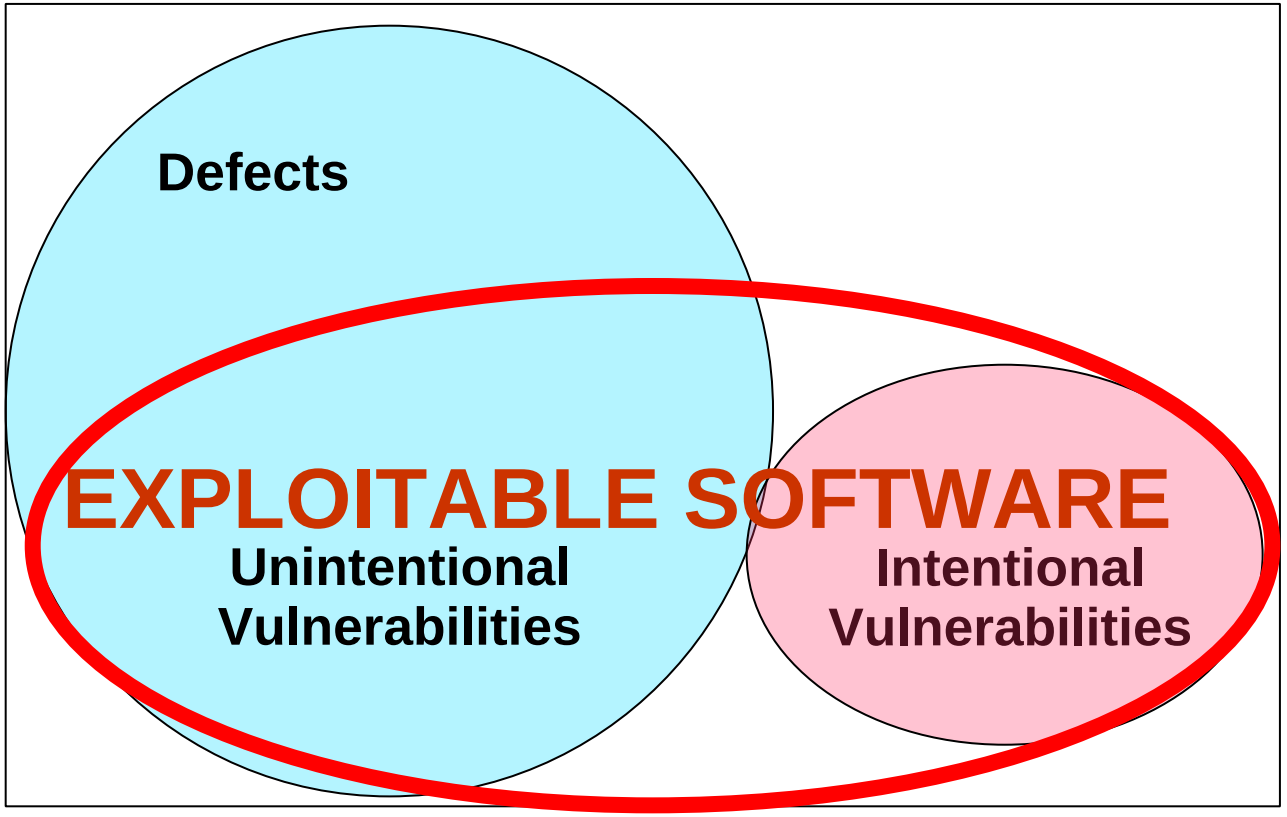

Security Feature
Buffer Overflow Exploit
SQL Injection Exploit

Exploitable Software Weaknesses (a.k.a. Vulnerabilities)

Vulnerabilities can be the outcome of non-secure practices and/or malicious intent of someone in the development/support lifecycle.

The exploitation potential of a vulnerability is independent of the "intent" behind how it was introduced.

DEPLOYED SOFTWARE



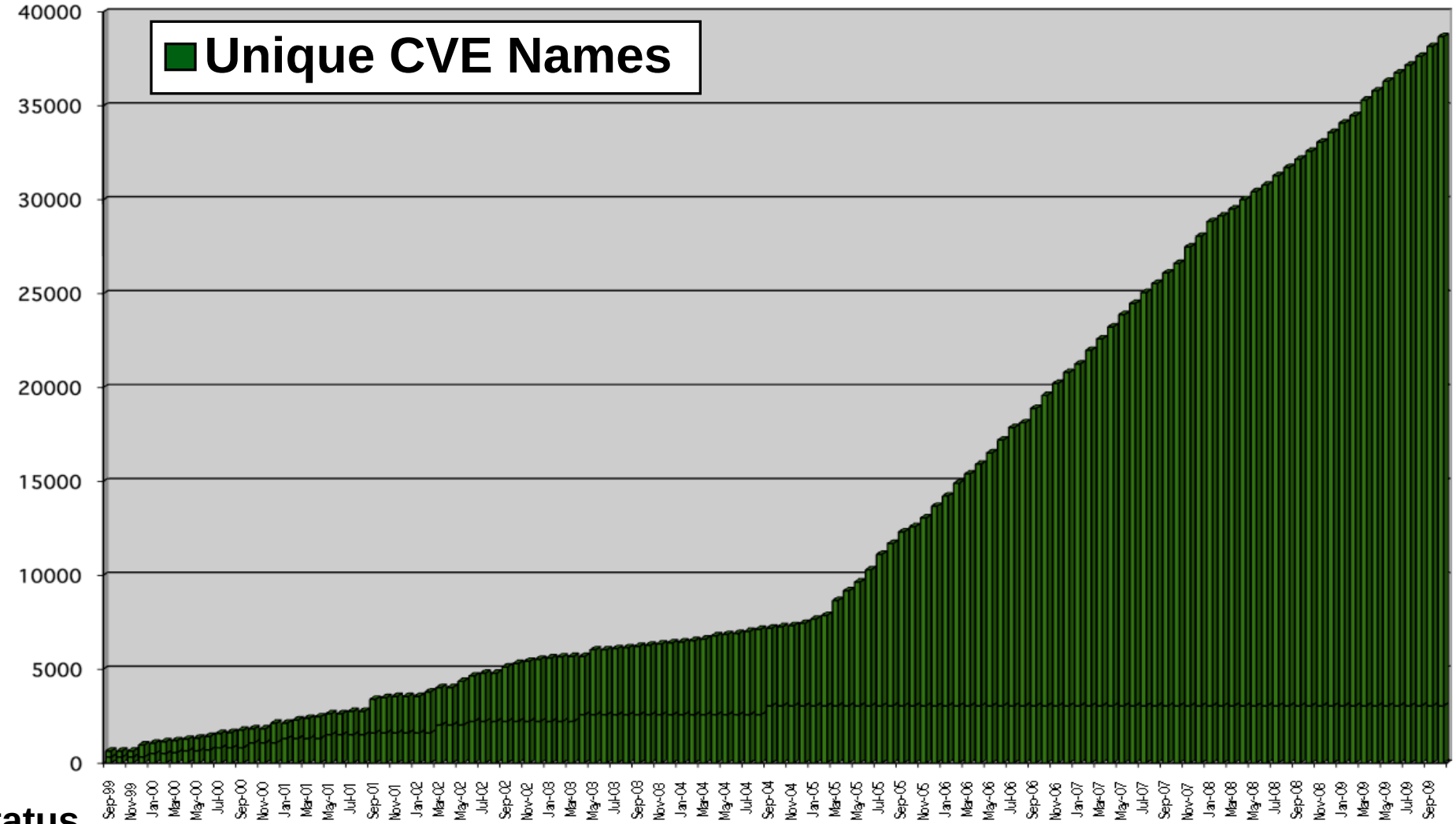
Intentional vulnerabilities are spyware & malicious logic deliberately imbedded (and might not be considered defects but they can make use of the same weakness patterns as unintentional mistakes)

Note: Chart is not to scale – notional representation -- for discussions

Software Vulnerabilities

- Serve as a primary point of entry that Attackers use to gain access to systems and/or data
- Expose business/mission systems to compromise
- Allow Attackers to circumvent security controls:
  - **Pose as other entities**
  - **Execute commands as other users**
  - **Conduct information gathering activities**
  - **Contrary to specified access restrictions**
    - Access and Manipulate data
  - **Hide activities**
  - **Conduct a denial of service**
  - **Embed malicious logic for future exploitation**

Publicly Known Vulnerabilities in "Packaged Software" (CVE) Growth

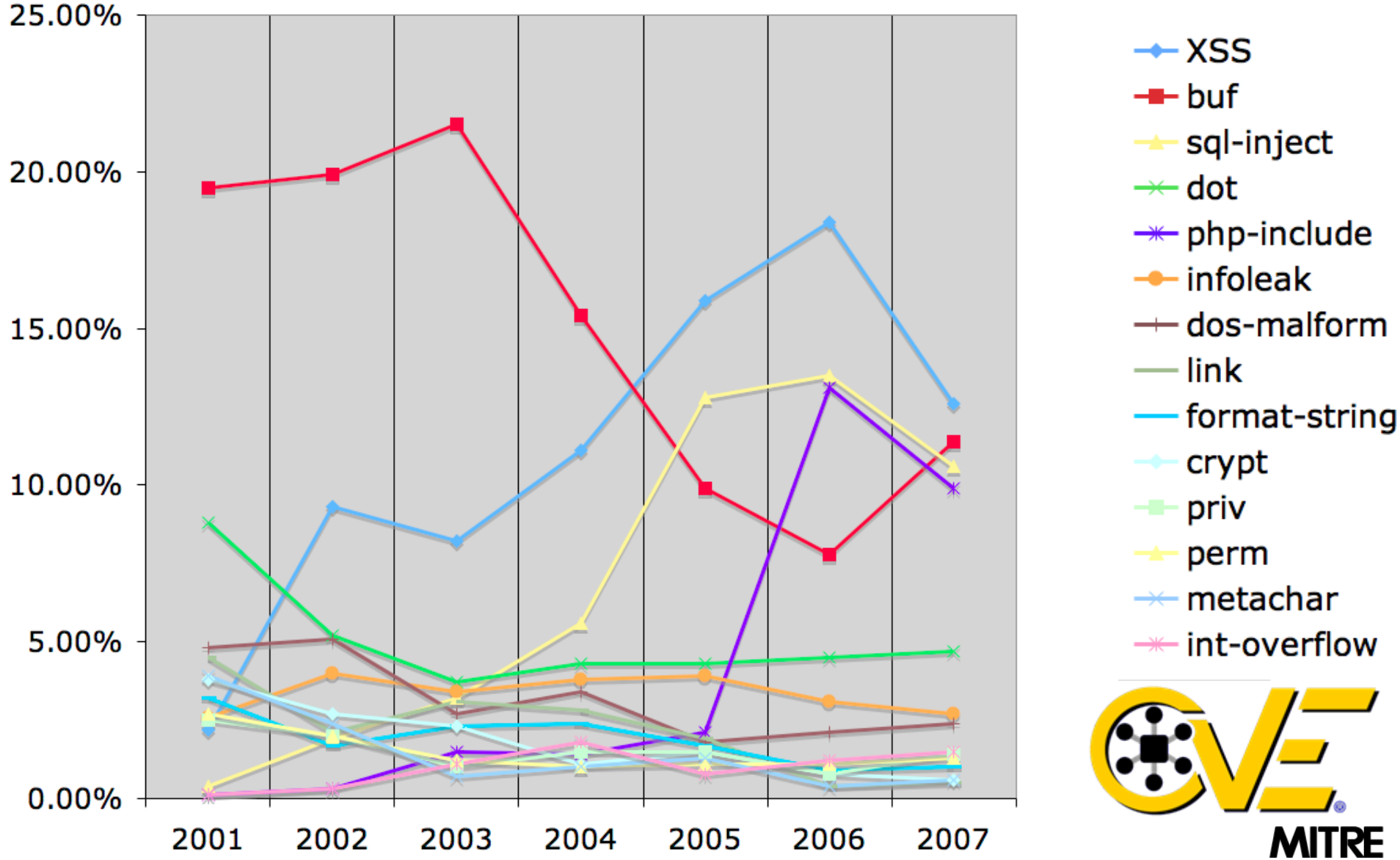


Status

(as of Oct 14, 2009)

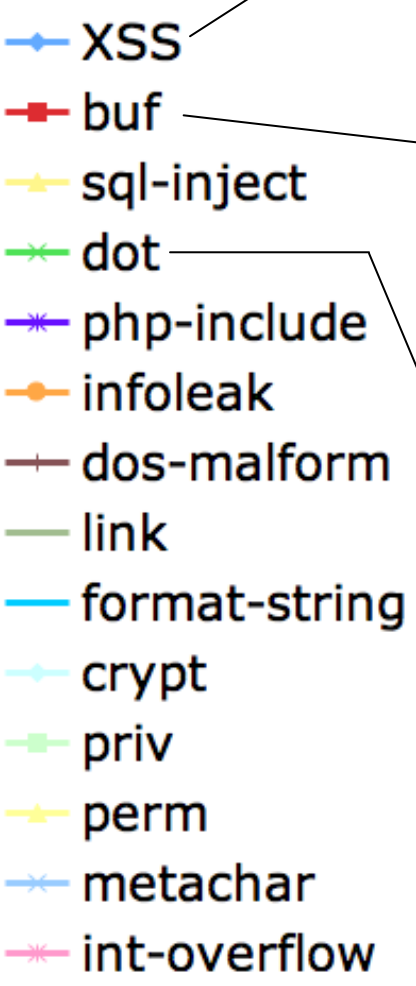
- **38,839 unique CVE names**

Vulnerability Type Trends: A Look at the CVE List (2001 - 2007)



CVE
MITRE

Removing and Preventing the Vulnerabilities Requires More Specific Definitions…CWEs



Failure to Sanitize Directives in a Web Page (aka 'Cross-site scripting' (XSS)) (79)
- Failure to Sanitize Script-Related HTML Tags in a Web Page (Basic XSS) (80)
- Failure to Sanitize Directives in an Error Message Web Page (81)
- Failure to Sanitize Script in Attributes of IMG Tags in a Web Page (82)
- Failure to Sanitize Script in Attributes in a Web Page (83)
- Failure to Resolve Encoded URI Schemes in a Web Page (84)
- Doubled Character XSS Manipulations (85)
- Invalid Characters in Identifiers (86)
- Alternate XSS syntax (87)

Failure to Constrain Operations within the Bounds of an Allocated Memory Buffer (119)
- Unbounded Transfer (‘Classic Buffer Overflow’) (120)
- Write-what-where Condition (123)
- Boundary Beginning Violation ('Buffer Underwrite') (124)
- Out-of-bounds Read (125)
- Wrap-around Error (128)
- Unchecked Array Indexing (129)
- Incorrect Calculation of Buffer Size (131)
- Miscalculated Null Termination (132)
- Return of Pointer Value Outside of Expected Range (466)

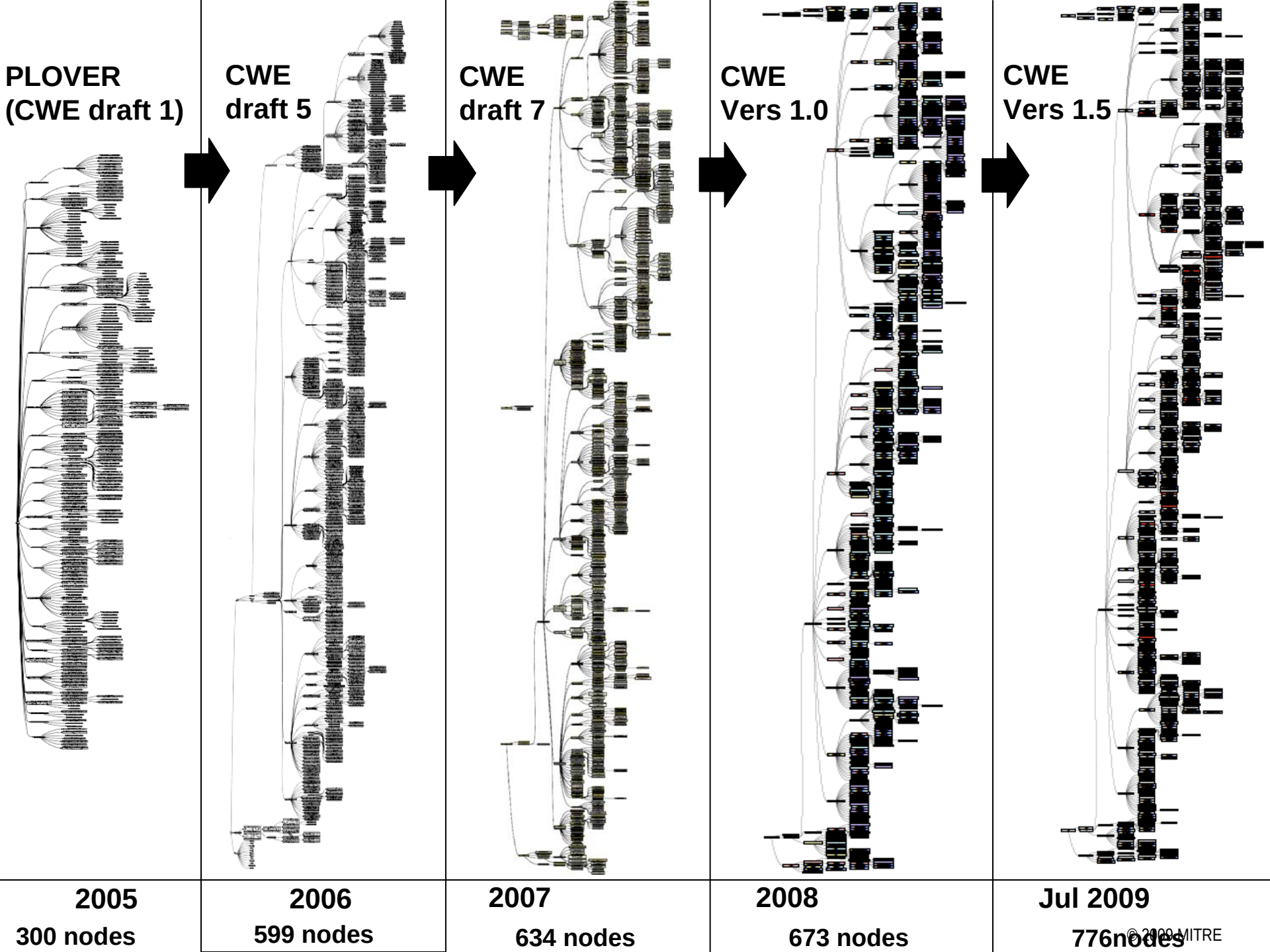
Path Traversal (22)
- Relative Path Traversal (23)
  - Path Traversal: '\..\filename' (29)
  - Path Traversal: '\dir\..\filename' (30)
  - Path Traversal: 'dir\..\filename' (31)
  - Path Traversal: '...' (Triple Dot) (32)
  - Path Traversal: '....' (Multiple Dot) (33)
  - Path Traversal: '....//' (34)
  - Path Traversal: '.../...//' (35)
- Absolute Path Traversal (36)
  - Path Traversal: '/absolute/pathname/here’ (37)
  - Path Traversal: '\absolute\pathname\here’ (38)
  - Path Traversal: 'C:dirname’ (39)
  - Path Traversal: '\\UNC\share\name\' (Windows UNC Share) (40)

Current Community Contributing to the Common Weakness Enumeration

- AppSIC
- Apple
- Aspect Security
- Booz Allen Hamilton Inc.
- Cenzic
- CERIAS/Purdue University
- CERT/CC
- Cigital
- Codenomicon
- Core Security
- Coverity
- DHS
- Fortify
- Gramma Tech
- IPA/JPCERT
- IBM
- Interoperability Clearing House
- JHU/APL
- JMU
- Kestrel Technology
- KDM Analytics
- Klocwork
- McAfee
- Microsoft
- MIT Lincoln Labs
- MITRE
- North Carolina State University
- NIST
- NSA
- OMG
- Oracle
- Ounce Labs
- OSD
- OWASP
- Palamida
- Parasoft
- PolySpace Technologies
- proServices Corporation
- SANS Institute
- SecurityInnovation
- Security University
- Semantic Designs
- SofCheck
- SPI Dynamics
- SureLogic, Inc.
- Symantec
- UNISYS
- VERACODE
- Watchfire
- WASC
- Whitehat Security, Inc.

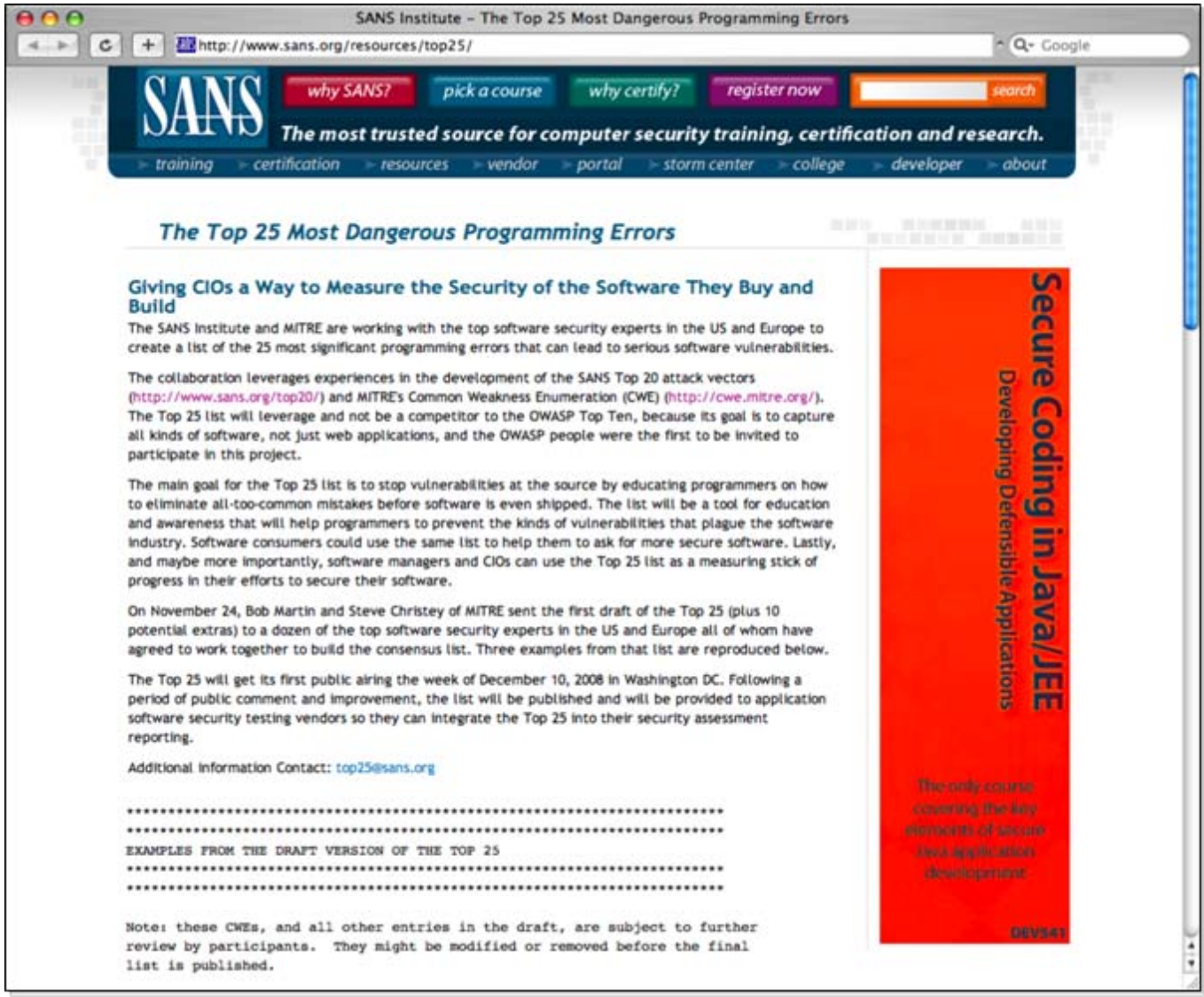
Making Security Measurable™

To join send e-mail to cwe@mitre.org


PLOVER
(CWE draft 1)
CWE
draft 5
CWE
draft 7
CWE
Vers 1.0
CWE
Vers 1.5
2005
300 nodes
2006
599 nodes
2007
634 nodes
2008
673 nodes
Jul 2009
776nodes

2009 SANS/CWE Top 25 Programming Errors (released 12 Jan 2009)

Some Participants:
Purdue
DHS
NSA
UC-Davis
KRvW Associates
Cigital
Symantec
McAfee
MITRE
Aspect Security
Secunia
Mandiant
Red Hat
Apple
Microsoft
Oracle
Fortify
Grammatech
Hatha Systems/KDM Analytics
Veracode
Breach Security



http://www.sans.org/top25errors/

Individual CAPEC Dictionary Definition (Release 1.2)

Blind SQL Injection

Attack Pattern ID 7

Pattern Abstraction: Detailed

Typical Severity High

Description

Summary

Blind SQL Injection results from an insufficient mitigation for SQL Injection. Although suppressing database error messages are considered best practice, the suppression alone is not sufficient to prevent SQL Injection. Blind SQL Injection is a form of SQL Injection that overcomes the lack of error messages. Without the error messages that facilitate SQL Injection, the attacker constructs input strings that probe the target through simple Boolean SQL expressions. The attacker can determine if the syntax and structure of the injection was successful based on whether the query was executed or not. Applied iteratively, the attacker determines how and where the target is vulnerable to SQL Injection.

In order to achieve this using Blind SQL Injection, an attacker:

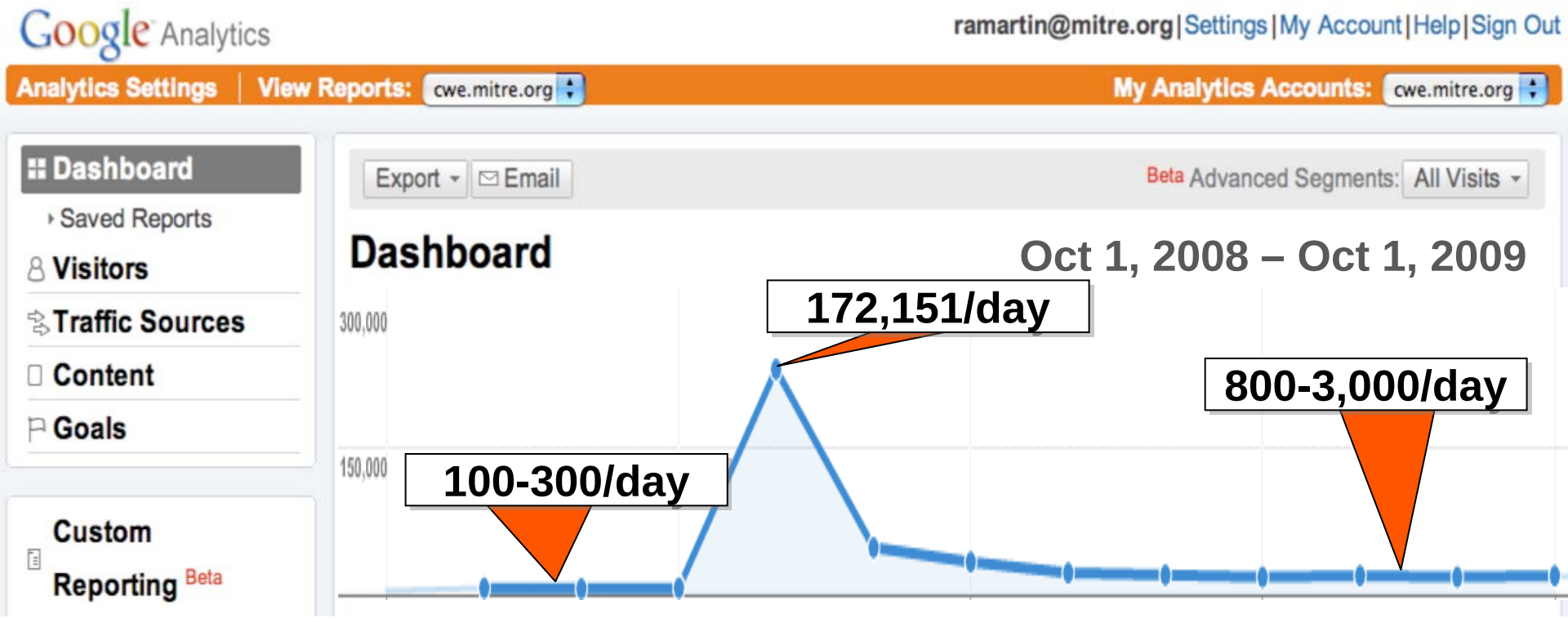
For example, an attacker may try entering something like "username' AND 1=1; --" in an input field. If the result is the same as when the attacker entered "username" in the field, then the attacker knows that the application is vulnerable to SQL Injection. The attacker can then ask yes/no questions from the database server to extract information from it. For example, the attacker can extract table names from a database using the following types of queries:

"username' AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 1, 1))) > 108".

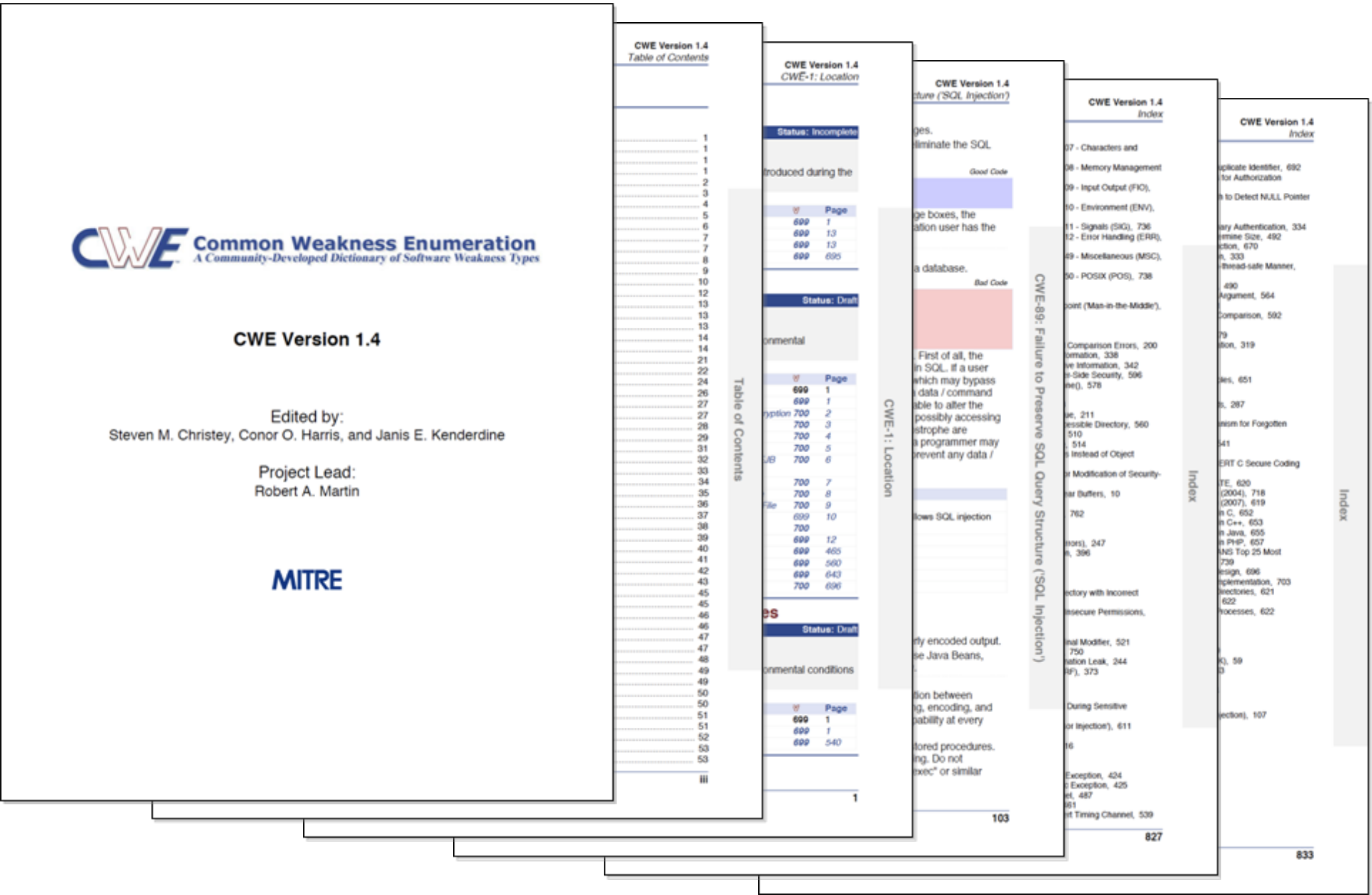
If the above query executes properly, then the attacker knows that the first character in a table name in the database is a letter between m and z. If it doesn't, then the attacker knows that the character must be between a and l (assuming of course that table names only contain alphabetic characters). By performing a binary search on all character positions, the attacker can determine all table names in the database. Subsequently, the attacker may execute an actual attack and send something like:

"username'; DROP TABLE trades; --

People are Starved for Simplicity



Printable PDFs of Entire CWE Now Available



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 starts; this is the correct fix for a TOCTOU bug - move the check as close as possible to the action because, in

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.

September 2008 (5)
August 2008 (2)
July 2008 (8)
June 2008 (4)

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

CWE Outreach: A Team Sport

May/June Issue of IEEE Security & Privacy…

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 re
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 secur
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/

SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

Fundamental Practices for Secure Software Development

A Guide to the Most Effective Secure Development Practices in Use Today

OCTOBER 8, 2008

Lead Writer Michael Howard, Microsoft Corp.

Contributors

Gunter Bitz, SAP AG
Jerry Cochran, Microsoft Corp.
Matt Coles, EMC Corporation
Danny Dhillon, EMC Corporation
Chris Fagan, Microsoft Corp.
Cassio Goldschmidt, Symantec Corp.
Wesley Higaki, Symantec Corp.
Steve Lipner, Microsoft Corp.
Brad Minnis, Juniper Networks, Inc.
Hardik Parekh, EMC Corporation
Dan Reddy, EMC Corporation
Alexandr Seleznyov, Nokia
Reeny Sondhi, EMC Corporation
Janne Uusilehto, Nokia
Antti Vähä-Sipilä, Nokia

CWE
CAPEC

SANS Common Security Errors in Programming

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 action 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

Special thanks to Robert A. Martin of Mitre Corporation.

Please send questions, comments and suggestions to spa@sans.org

Top 7 Secure Coding Web Sites

1. CERT Secure Coding Standards www.securecoding.cert.org
2. Secure Coding www.cert.org/secure-coding
3. JTC1/SC22/WG14 - C www.open-std.org/jtc1/sc22/wg14
4. JTC1/SC22/WG21 - The C++ Standards Committee www.open-std.org/jtc1/sc22/wg21
5. C Language Gotchas www.greenend.org.uk/rjk/2001/02/cfu.html
6. NIST SAMATE Reference Dataset Project samate.nist.gov/SRD
7. comp.lang.c Frequently Asked Questions c-faq.com

http://www.securecoding.cert.org

How to make sure programmers can avoid these errors: Sign up for the SANS Secure Programming Assessment Program at www.sans-ssi.org.

SOURCE CODE

ERROR HANDLING
- ERROR CONDITIONS, RETURN VALUES, STATUS CODES
- MISSING ERROR HANDLING MECHANISM
- MISSING CATCH BLOCK

API ABUSE
- HEAP INSPECTION
- CALL TO LIMITED API
- DANGEROUS FUNCTIONS
- MULTIPLE BINDS TO SAME PORT
- STRUCTURE AND VALIDITY PROBLEMS
- J2EE BAD PRACTICES: GETCONNECTION()
- MISINTERPRETED FUNCTION RETURN VALUE
- OFTEN MISUSED: ARGUMENTS AND PARAMETERS
- OFTEN MISUSED: AUTHENTICATION
- OFTEN MISUSED: EXCEPTION HANDLING
- OFTEN MISUSED: PATH MANIPULATION
- OFTEN MISUSED: PRIVILEGE MANAGEMENT
- OFTEN MISUSED: STRING MANAGEMENT
- J2EE BAD PRACTICES: SOCKETS
- UNCHECKED RETURN VALUE
- UNSAFE URL REDIRECTION
- DIRECTORY RESTRICTION
- REACHABLE ASSERTION

ENCAPSULATION
- MOBILE CODE ISSUES
- LEFTOVER DEBUG CODE
- DYNAMIC CLASS LOADING
- ERRONEOUS CLONE METHOD
- COMPARING CLASSES BY NAME
- DATA LEAKING BETWEEN USERS
- TRUSTED BOUNDARY VIOLATION
- RELYING ON PACKAGE-LEVEL SCOPE
- PERSISTENCE IN J2EE FRAMEWORKS
- DESERIALIZATION OF UNTRUSTED DATA
- OVERFLOW OF STATIC INTERNAL BUFFER
- INFORMATION LEAK THROUGH SERIALIZATION
- INFORMATION LEAK THROUGH CLASS CLONING
- PUBLIC DATA ASSIGNED TO PRIVATE ARRAY-TYPED FIELD
- PRIVATE ARRAY-TYPED FIELD RETURNED FROM A PUBLIC METHOD

TIME AND STATE
- STATE ISSUES
- RACE CONDITIONS
- SESSION FIXATION
- CONCURRENCY ISSUES
- TEMPORARY FILE ISSUES
- COVERT TIMING CHANNEL
- TECHNOLOGY-SPECIFIC TIME AND STATE ISSUES
- SYMBOLIC NAME NOT MAPPING TO CORRECT OBJECT

SECURITY FEATURES
- HARD-CODED PASSWORD (259)
- UNVERIFIED PASSWORD CHANGE
- EMPTY PASSWORD IN CONFIGURATION FILE
- MISSING PASSWORD FIELD MASKING
- WEAK CRYPTOGRAPHY FOR PASSWORDS
- WEAK PASSWORD REQUIREMENTS
- NOT ALLOWING PASSWORD AGING
- ALLOWING UNCHECKED PASSWORD AGING
- CREDENTIALS MANAGEMENT
- INSUFFICIENTLY PROTECTED CREDENTIALS (522)
- PRIVACY VIOLATION
- ORIGIN VALIDATION ERROR
- IMPROPERLY VERIFIED SIGNATURE
- USE OF LESS TRUSTED SOURCE
- UNTRUSTED DATA APPENDED WITH TRUSTED DATA
- IMPROPERLY TRUSTED REVERSE DNS
- INSUFFICIENT TYPE DISTINCTION
- INSUFFICIENT VERIFICATION OF DATA
- USE OF COOKIES
- CROSS-SITE REQUEST FORGERY (352)
- FAILURE TO ADD INTEGRITY CHECK VALUE
- FAILURE TO CHECK INTEGRITY CHECK VALUE
- TRUST OF SYSTEM EVENT DATA
- FAILURE TO ENCRYPT DATA (311)
- PLAINTEXT STORAGE OF SENSITIVE INFORMATION (312)
- KEY MANAGEMENT ERRORS (320 [321])
- MISSING REQUIRED CRYPTOGRAPHIC STEP (325)
- WEAK ENCRYPTION (326)
- REVERSIBLE ONE-WAY HASH (328)
- NOT USING A RANDOM IV WITH CBC MODE (329)
- CRYPTOGRAPHIC ISSUES (310)
- CLIENT-SIDE ENFORCEMENT OF SERVER-SIDE SECURITY
- IMPROPERLY IMPLEMENTED SECURITY CHECK FOR STANDARD
- AUTHENTICATION ISSUES (287 [288, 301])
- USER INTERFACE SECURITY ISSUES
- RANDOMNESS AND PREDICTABILITY
- PERMISSIONS, PRIVILEGES, & ACCESS CONTROLS
- ACCESS CONTROL ISSUES (285)
- PERMISSION ISSUES (275)

CODE QUALITY
- RESOURCE EXHAUSTION
- MEMORY LEAK
- RESOURCE LEAK
- IMPROPER RESOURCE SHUTDOWN OR RELEASE
- ASYMMETRIC RESOURCE CONSUMPTION
- INSUFFICIENT RESOURCE POOL
- RESOURCE MANAGEMENT ERRORS
- DOUBLE FREE (415)
- EXPLICIT CALL TO FINALIZE
- IMPROPERLY FREEING HEAP MEMORY
- ERRONEOUS FINALIZE METHOD
- USE AFTER FREE
- WEB PARAMETER TAMPERING (472)
- MODIFICATION OF ASSUMED-IMMUTABLE DATA
- PHP EXTERNAL VARIABLE MODIFICATION
- PUBLIC STATIC FINAL FIELD REFERENCES MUTABLE OBJECT
- NULL POINTER DEREFERENCE (476)
- FAILURE TO FOLLOW EXECUTION
- INCORRECT BLOCK DELIMITATION
- OMITTED BREAK STATEMENT
- UNSPECIFIED BEHAVIOR
- UNTRUSTED SEARCH PATH (426)
- CHANNEL AND PATH ERRORS
- ALTERNATE PATH ERRORS (425)
- CHANNEL ERRORS
- UNPROTECTED WINDOWS MESSAGING CHANNEL (422)
- UNPROTECTED ALTERNATE CHANNEL
- PROTECTED TRUSTED CHANNEL
- UNPROTECTED PRIMARY CHANNEL
- ALTERNATE CHANNEL RACE CONDITION
- HTTP RESPONSE SPLITTING
- UNPARSED RAW WEB CONTENT DELIVERY
- DANGEROUS HANDLER NOT CLEARED/DISABLED DURING SENSITIVE OPERATIONS
- UNRESTRICTED FILE UPLOAD (434)
- HANDLER ERRORS
- INCOMPLETE IDENTIFICATION OF UPLOADED FILE VARIABLES (PHP)
- FAILURE TO ACCOUNT FOR DEFAULT CASE IN SWITCH
- IMPROPER HANDLER DEPLOYMENT
- MISSING HANDLER
- SECURITY FEATURE/CONFIG/INIT ERRORS
- INSUFFICIENT ENTROPY
- INSUFFICIENT ENTROPY IN PRNG
- HARD-CODED CRYPTOGRAPHIC KEY
- INCORRECT PERMISSION ASSIGNMENT FOR CRITICAL RESOURCE
- USE OF A BROKEN OR RISKY CRYPTOGRAPHIC ALGORITHM
- INSUFFICIENT PSEUDO-RANDOM NUMBER SEED
- USE OF RANDOM VALUES FOR SECURITY
- INSUFFICIENT RANDOM VALUES
- USE OF PSEUDO-RANDOM NUMBER GENERATOR WITH PREDICTABLE SEED
- REFLECTION
- DANGEROUS FUNCTION CALL FROM A SIGNAL HANDLER
- INITIALIZATION AND CLEANUP ERRORS
- USER INTERFACE QUALITY ERRORS
- SUSPICIOUS COMMENT
- STACK ALLOCATION ARGUMENT
- POINTER ISSUES
- OBSOLETE
- EXPRESSION ISSUES
- INCONSISTENT IMPLEMENTATIONS
- DATA STRUCTURE ISSUES
- UNUSED VARIABLE
- DEAD CODE

DATA HANDLING
- INTEGER OVERFLOW (190)
- INTEGER UNDERFLOW
- INTEGER COERCION ERROR
- OFF-BY-ONE ERROR
- SIGN EXTENSION ERROR
- SIGNED TO UNSIGNED CONVERSION ERROR (195)
- UNSIGNED TO SIGNED CONVERSION ERROR (196)
- NUMERIC TRUNCATION ERROR
- NUMERIC BYTE ORDERING ERROR
- NUMERIC ERRORS
- PATH TRAVERSAL (22)
- PATHNAME TRAVERSAL AND EQUIVALENCE ERRORS
- PATH EQUIVALENCE
- LINK FOLLOWING (59)
- VIRTUAL FILES
- PATH MANIPULATION
- PROCESS CONTROL
- MISSING XML VALIDATION
- INPUT VALIDATION
- COMMAND INJECTION (77 [78])
- CROSS-SITE SCRIPTING (79)
- SQL INJECTION (89)
- LDAP INJECTION (90)
- XML INJECTION (91)
- CRLF INJECTION (93)
- FORMAT STRING VULNERABILITY (134)
- SPECIAL ELEMENT INJECTION
- ARGUMENT INJECTION OR MODIFICATION
- CUSTOM SPECIAL CHARACTER INJECTION
- RESOURCE INJECTION
- TECHNOLOGY-SPECIFIC INPUT VALIDATION PROBLEMS
- MISINTERPRETATION OF INPUT
- UNCHECKED INPUT FOR LOOP CONDITION
- NULL BYTE INTERACTION ERROR (626)
- CODE INJECTION
- DYNAMIC VARIABLE EVALUATION
- VARIABLE EXTRACTION ERROR
- PHP FILE INCLUSION (98)
- DIRECT STATIC CODE INJECTION
- DIRECT DYNAMIC CODE EVALUATION (95)
- ENCODING ERROR
- CASE SENSITIVITY
- EARLY VALIDATION ERRORS
- VALIDATE-BEFORE CANONICALIZE
- VALIDATE-BEFORE FILTER
- COLLAPSE OF DATA INTO UNSAFE VALUE
- PERMISSIVE WHITELIST
- REPRESENTATION ERRORS
- INCOMPLETE BLACKLIST (184)
- REGULAR EXPRESSION ERROR
- PARTIAL COMPARISON
- INFORMATION LEAK (200)
- GENERAL INFORMATION MANAGEMENT PROBLEMS
- INFORMATION LOSS OR OMISSION
- CONTAINMENT ERRORS
- INFORMATION MANAGEMENT ERRORS
- BOUNDARY ERRORS
- TYPE ERRORS
- OUTPUT VALIDATION
- UNBOUNDED TRANSFER 'CLASSIC BUFFER OVERFLOW' (120)
- WRITE-WHAT-WHERE CONDITION (123)
- BOUNDARY BEGINNING VIOLATION (124)
- OUT-OF-BOUNDS READ (125)
- WRAP-AROUND ERROR (128)
- BUFFER ERRORS (119)
- UNCHECKED ARRAY INDEXING (129)
- LENGTH PARAMETER INCONSISTENCY (130)
- OTHER LENGTH CALCULATION ERROR (131)
- MISCALCULATED NULL TERMINATION (132)
- STRING ERRORS (133)

OWASP Top 10/Top CVE Causes | OWASP Top 10 | 2006 Top CVE Causes | Other CWE's

Major contributors to this project:

SANS MITRE CERT Software Engineering Institute Carnegie Mellon OWASP The Open Web Application Security Project

Common Security Errors in Programming

The SANS Common Security Errors in Programming map illustrates those software weaknesses that are responsible for the majority of the publicly known vulnerabilities reported in the Common Vulnerabilities and Exposures (CVE) and discovered in self-developed applications. 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 action to find those 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 CWE IDs for each weakness on the "2009 CWE/SANS Top 25 Most Dangerous Programming Errors" list. CWE IDs can be found at the MITRE CWE Web site or accessed directly by putting the number (in place of ###) in the following URL: http://cwe.mitre.org/data/definitions/###.html

Special thanks to the CWE Team at MITRE.

SANS MITRE

- 2009's CWE/SANS Top 25 Most Dangerous Programming Errors
- 2009's Common Programming Errors

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)

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

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

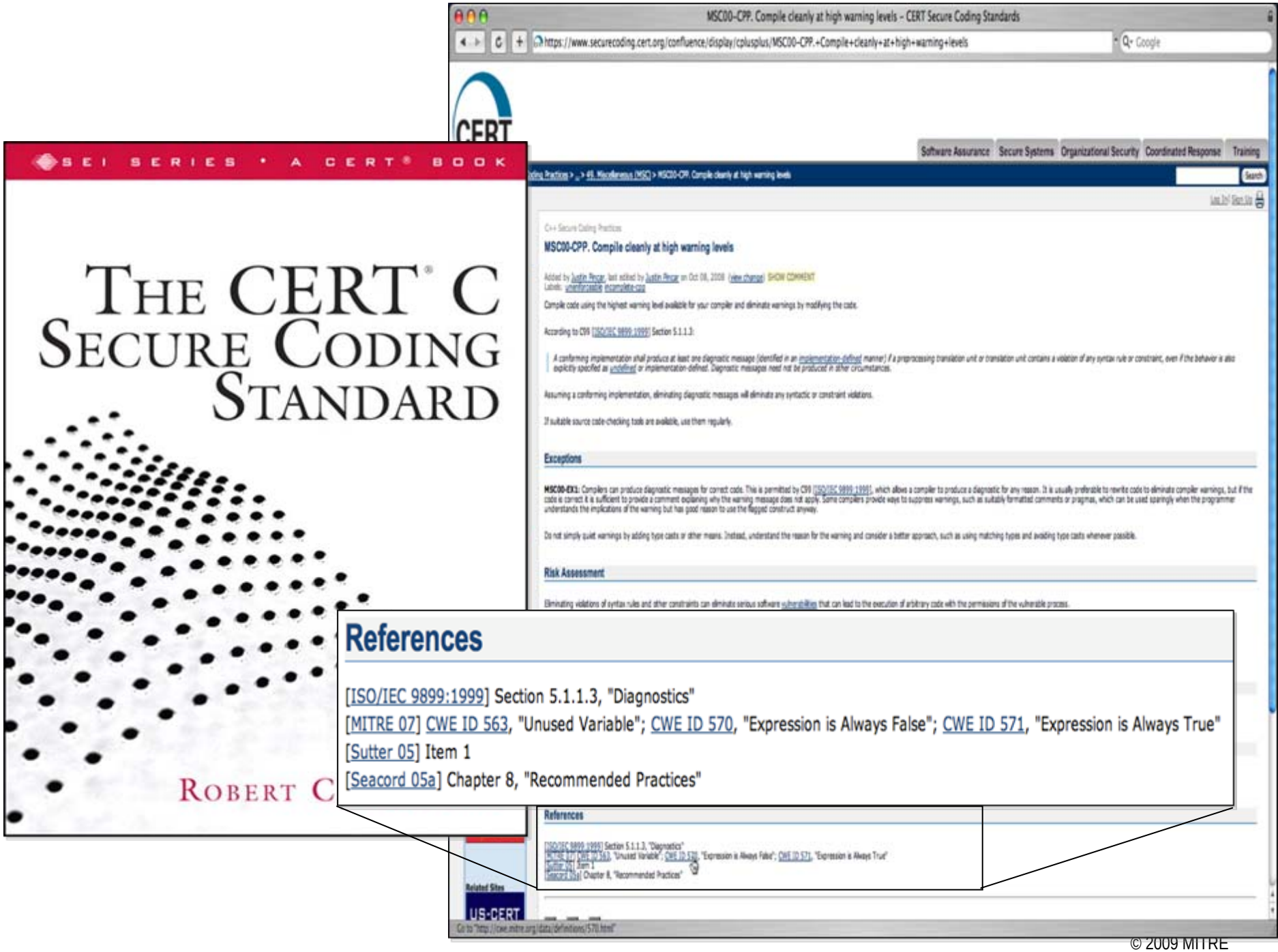
- NULL Pointer Dereference
- Incorrect Block Delimitation
- Omitted Break Statement in Switch
- Undefined Behavior for Input to API
- Use of Hard-coded, Security-relevant Constants
- Unsafe Function Call from a Signal Handler
- Suspicious Comment
- Return of Stack Variable Address
- Missing Default Case 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

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 - (284)
  - 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
  - Incorrect 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

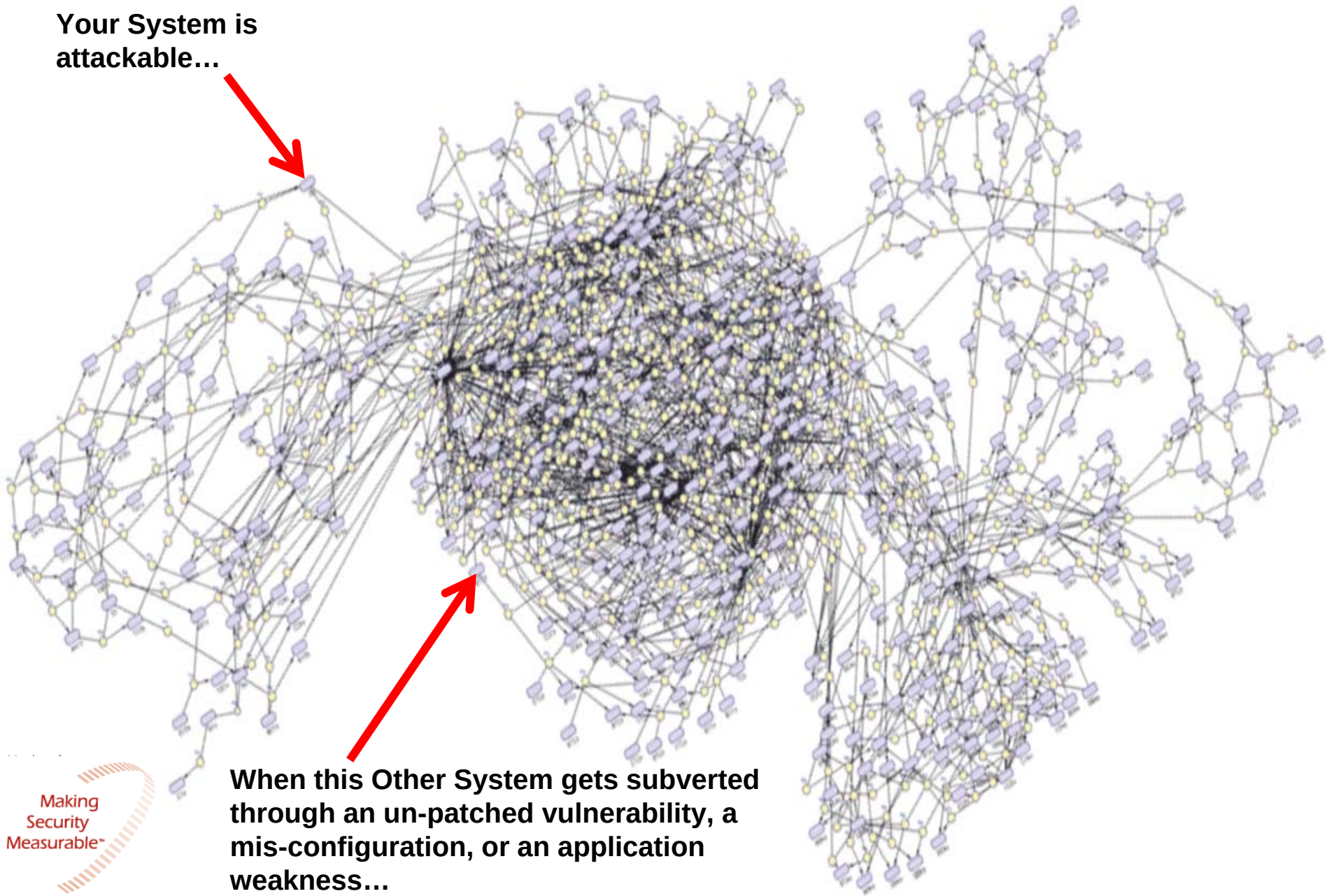
Insufficient Encapsulation

- 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


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
CERT
Software Assurance
Secure Systems
Organizational Security
Coordinated Response
Training
C++ Secure Coding Practices
MSC00-CPP. Compile cleanly at high warning levels
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:
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
Risk Assessment
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

Today Everything's Connected

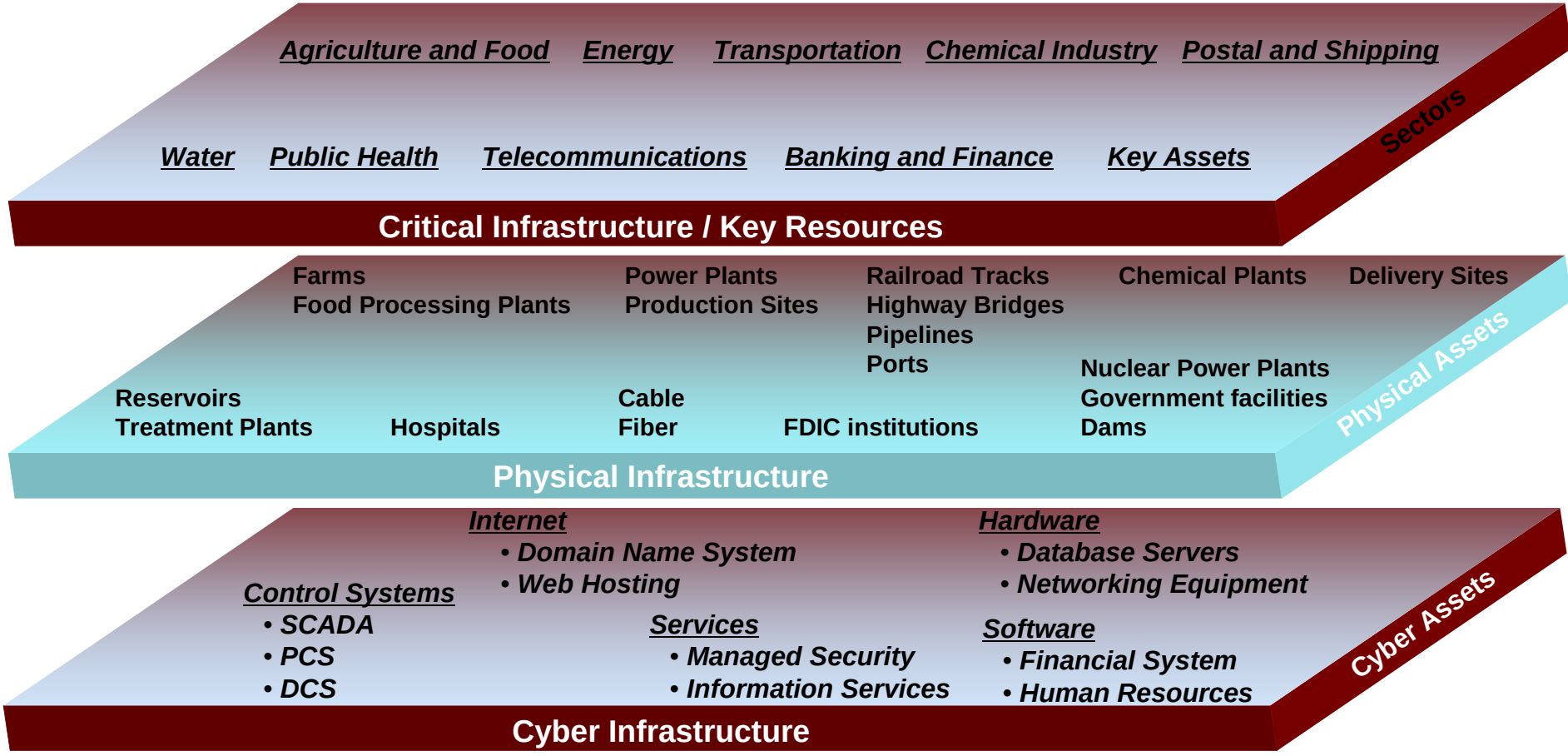
Your System is attackable…



When this Other System gets subverted through an un-patched vulnerability, a mis-configuration, or an application weakness…

Making Security Measurable™

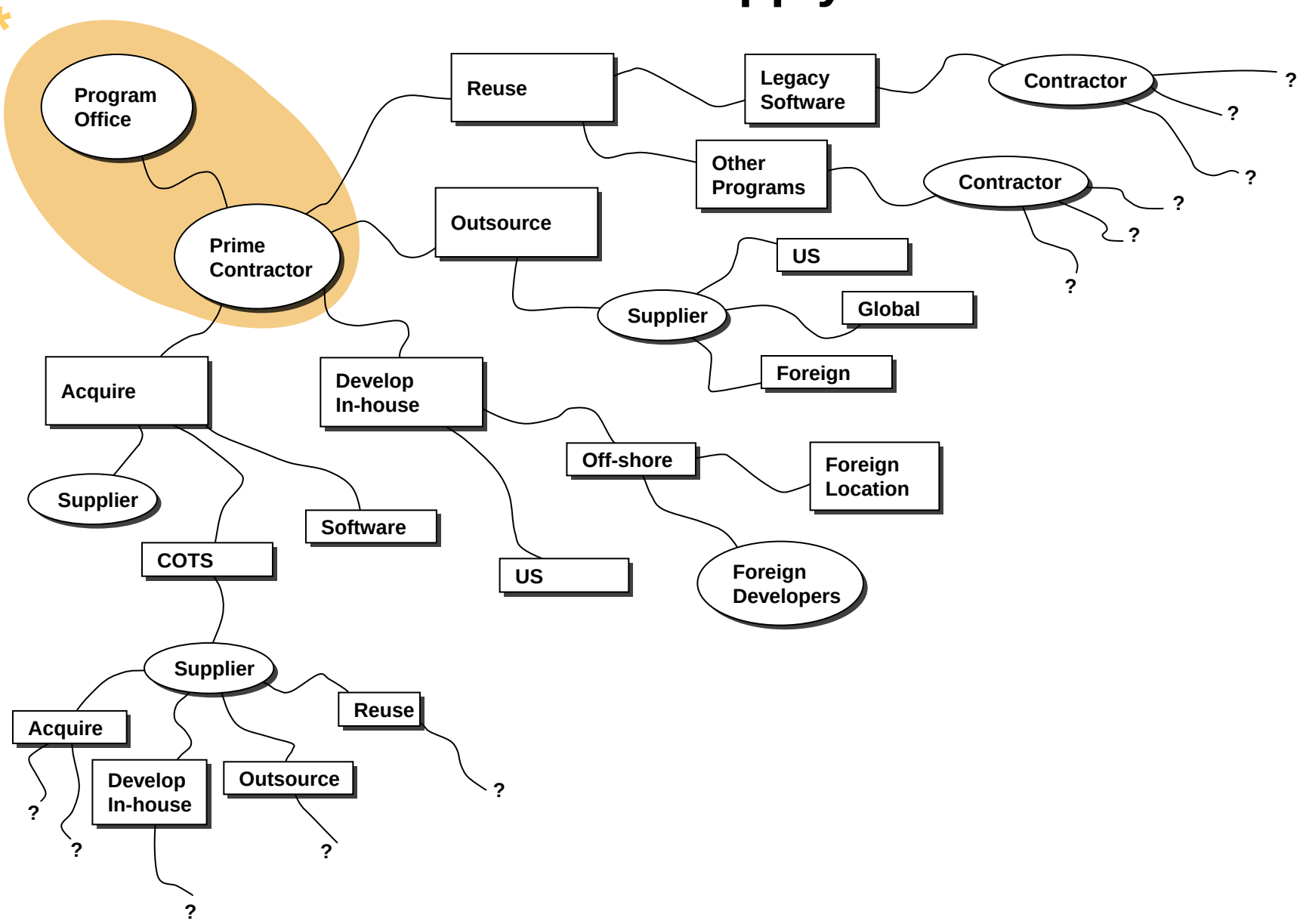
Cyberspace & physical space are increasingly intertwined and software controlled/enabled



Need for secure software applications

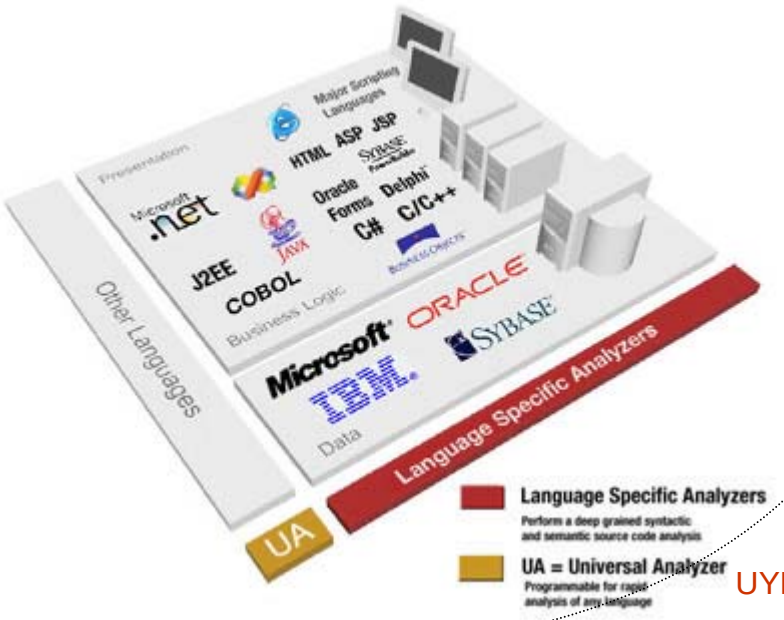
"In an era riddled with asymmetric cyber attacks, claims about system reliability, integrity and safety must also include provisions for built-in security of the enabling software."

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"

Our Systems are Composed of Elements from Many Languages and Environments



2GLs

…

SNX360

UYK20

BAL

COBOL

FORTRAN

3GLs

…

Pascal

BASIC

PL/1

C

SQL

Python

C++

Java/J2EE

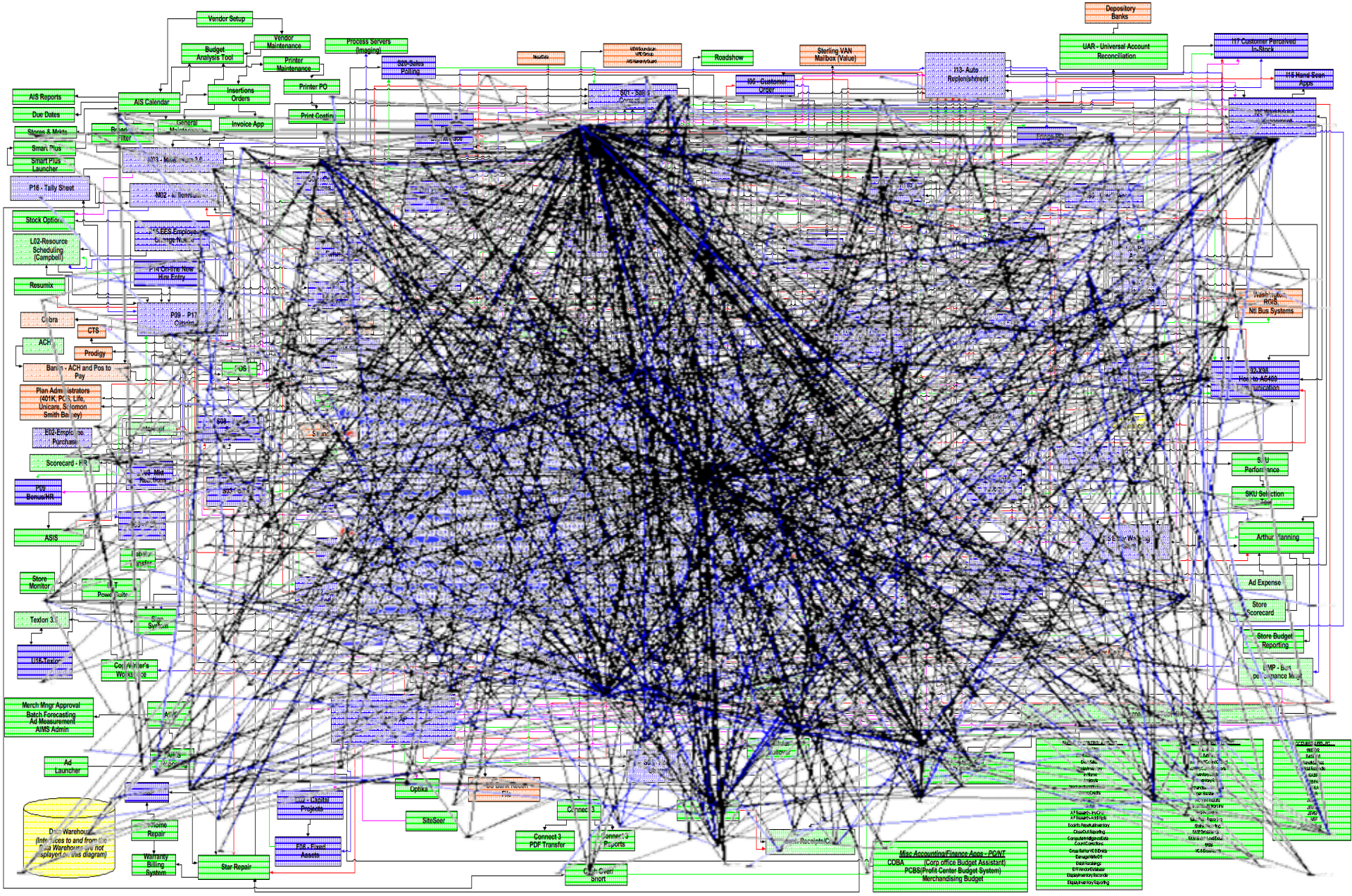
4GLs

…

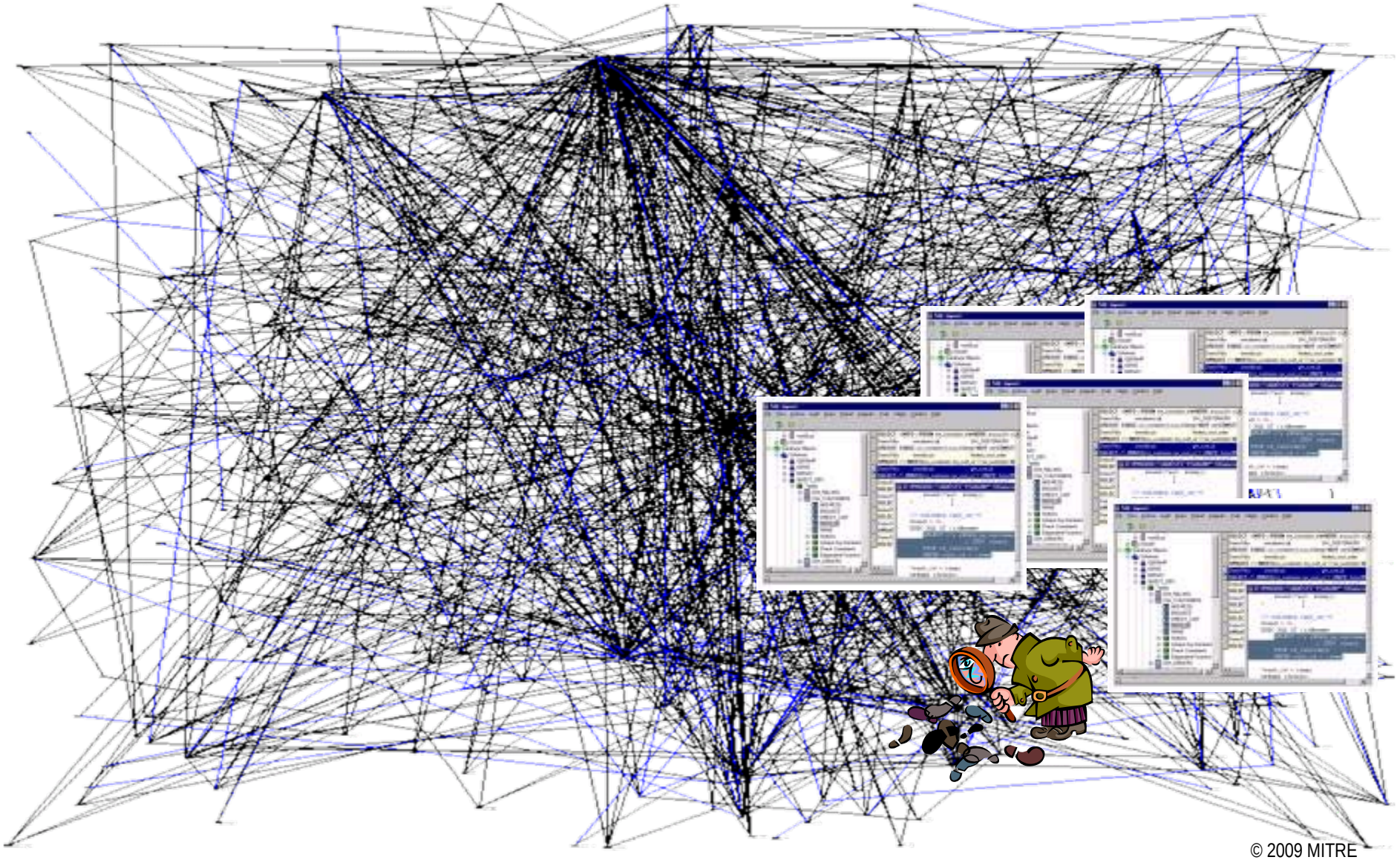
Natural

VisualBasic

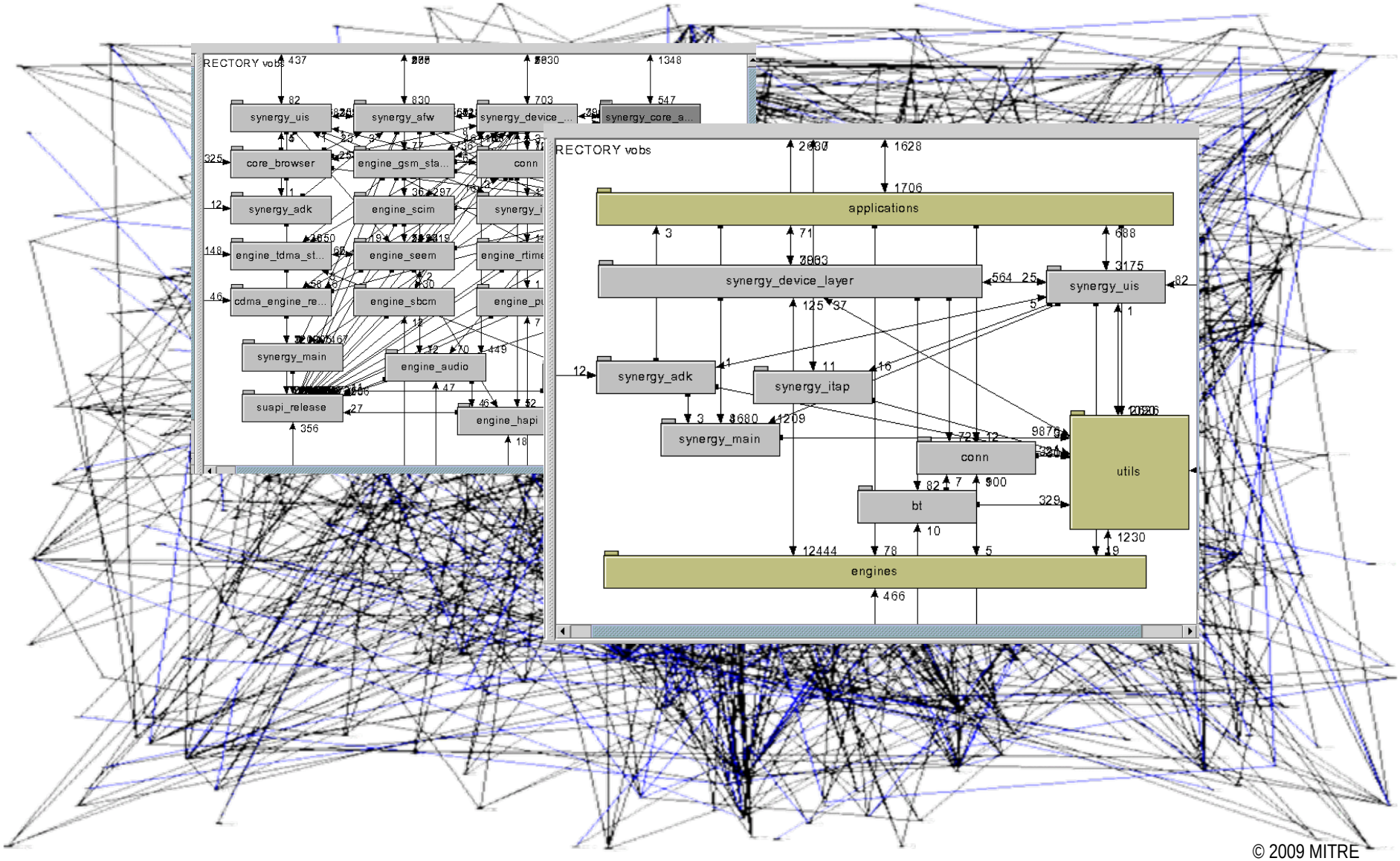
Systems Are Complicated…



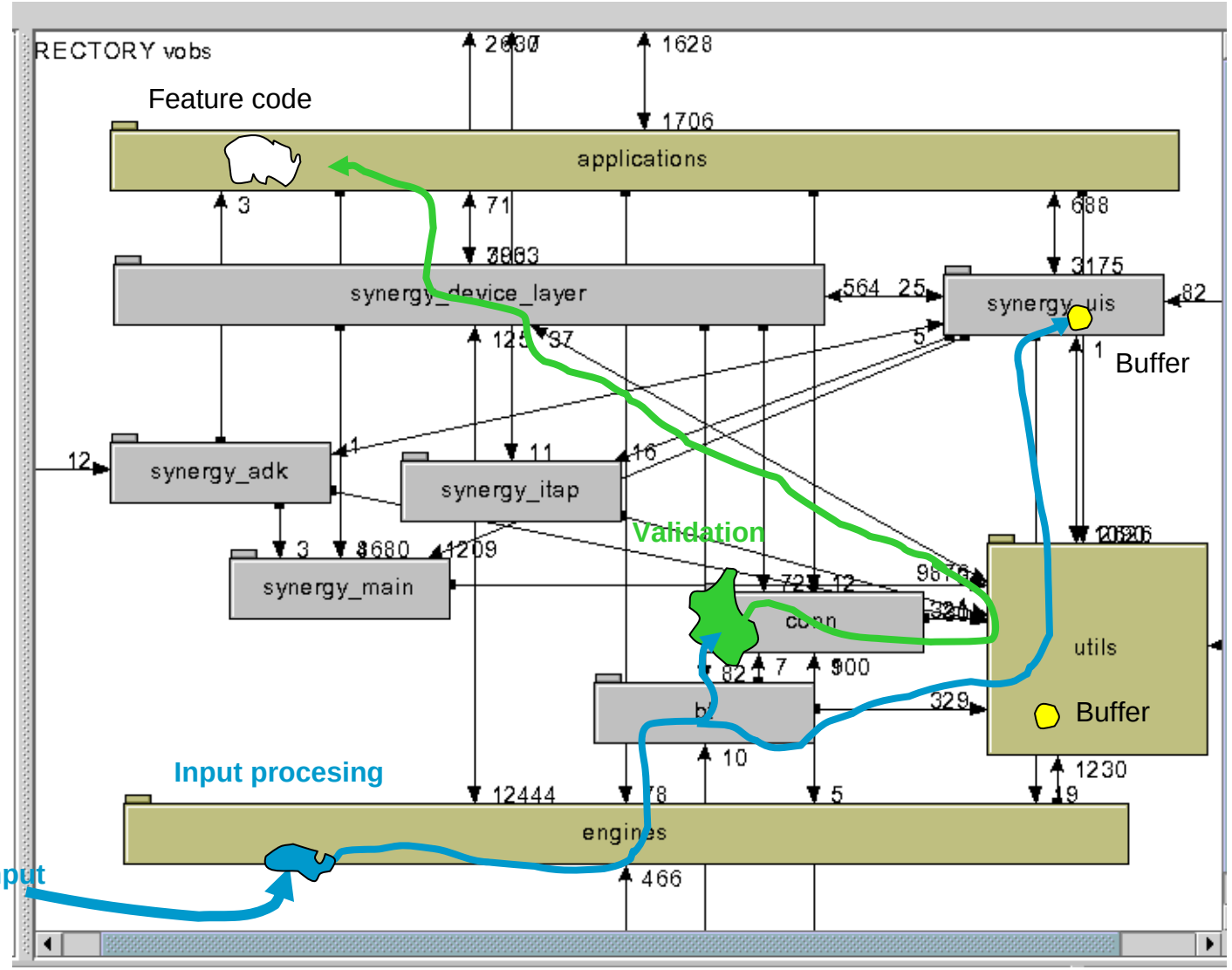
And Software Is Complex Too...

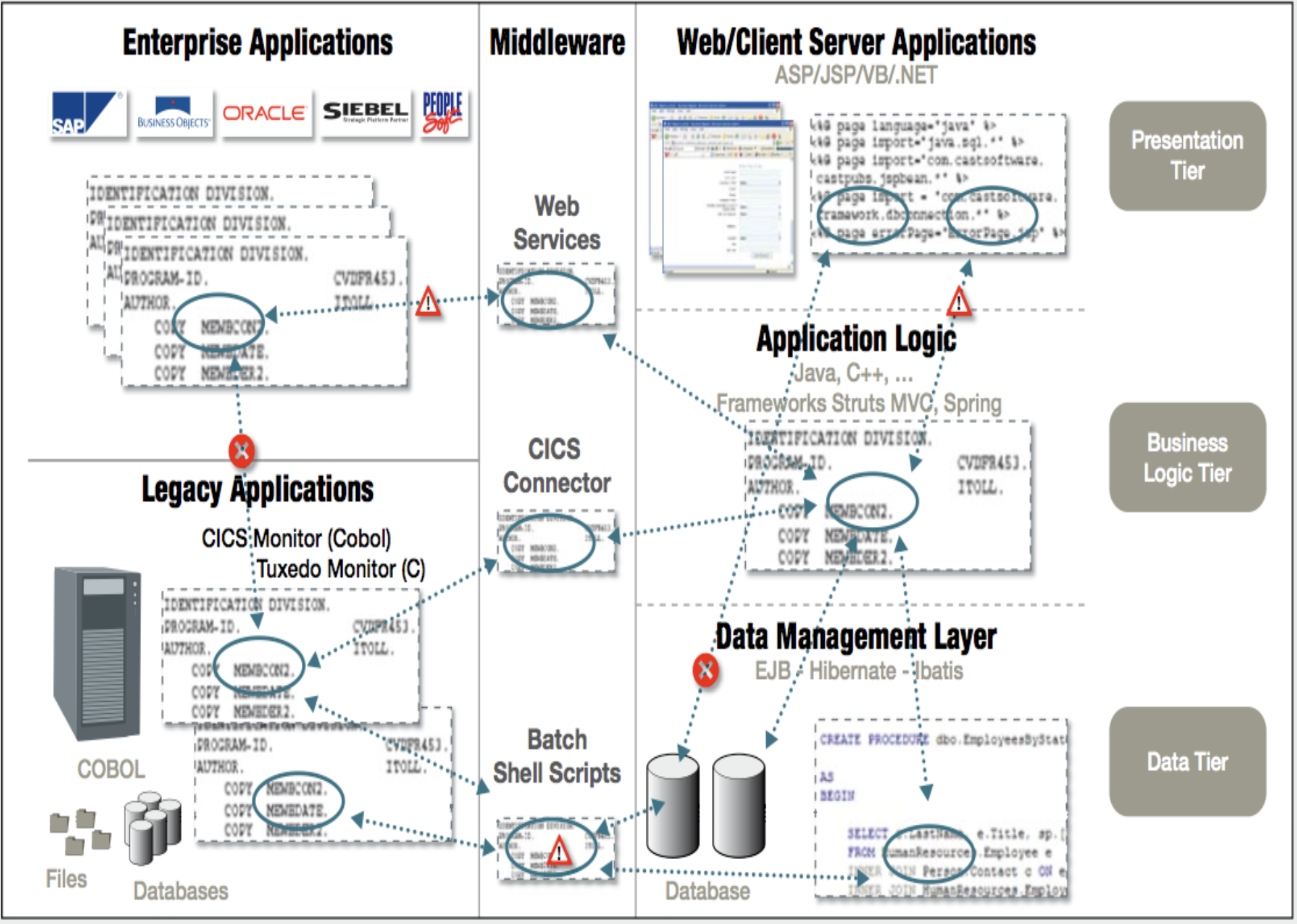


Some Static Analysis Tools Focus on Pulling Structure Out of the Complexity …

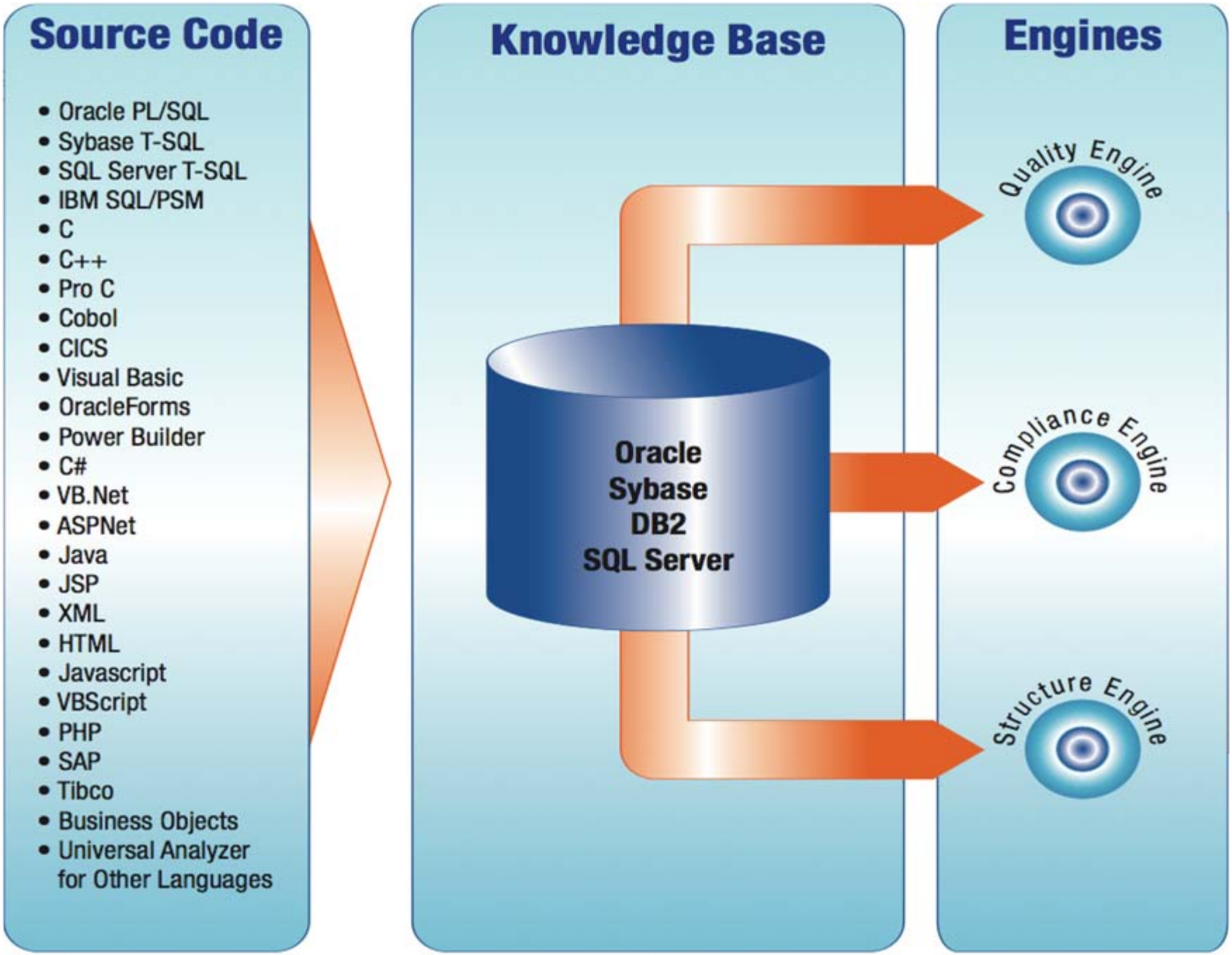


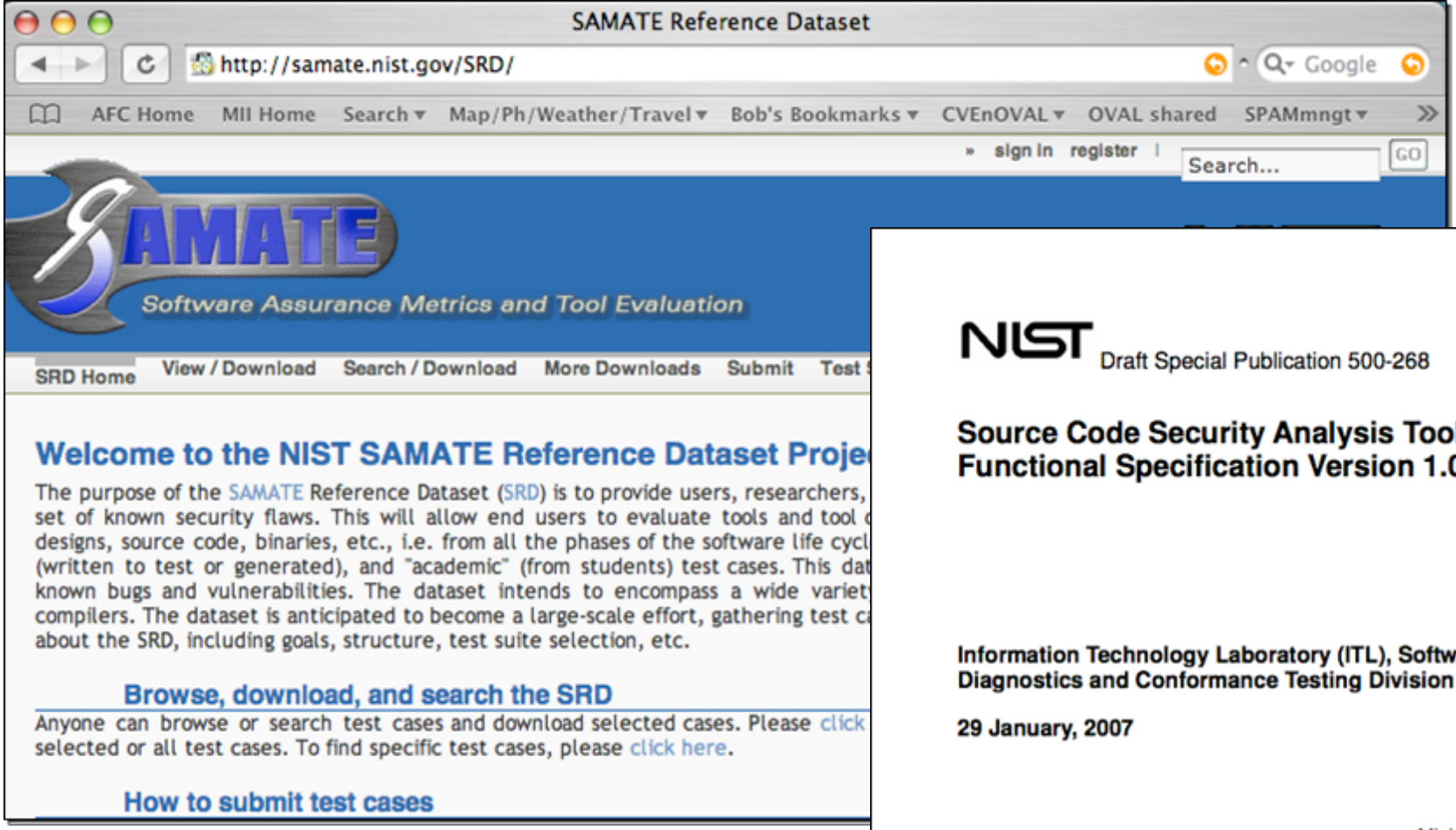
Static Analysis is about collecting information and capturing knowledge

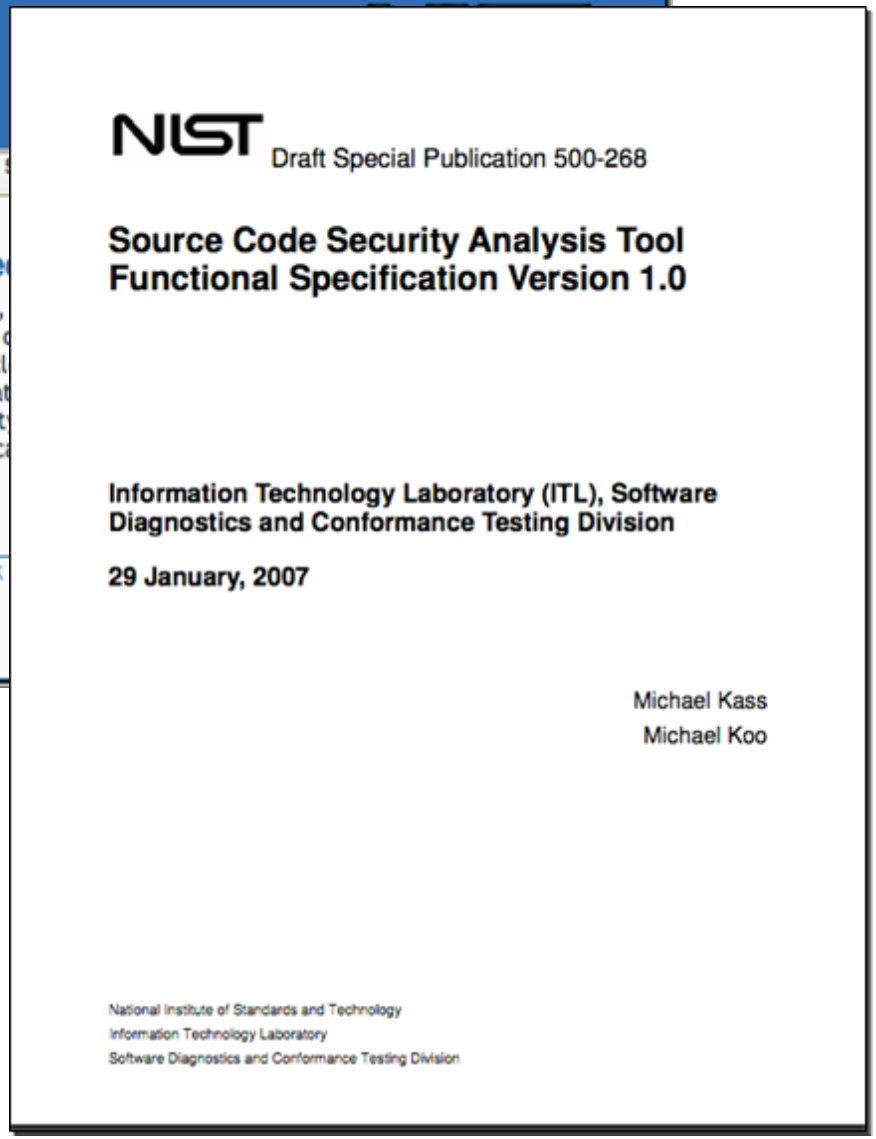


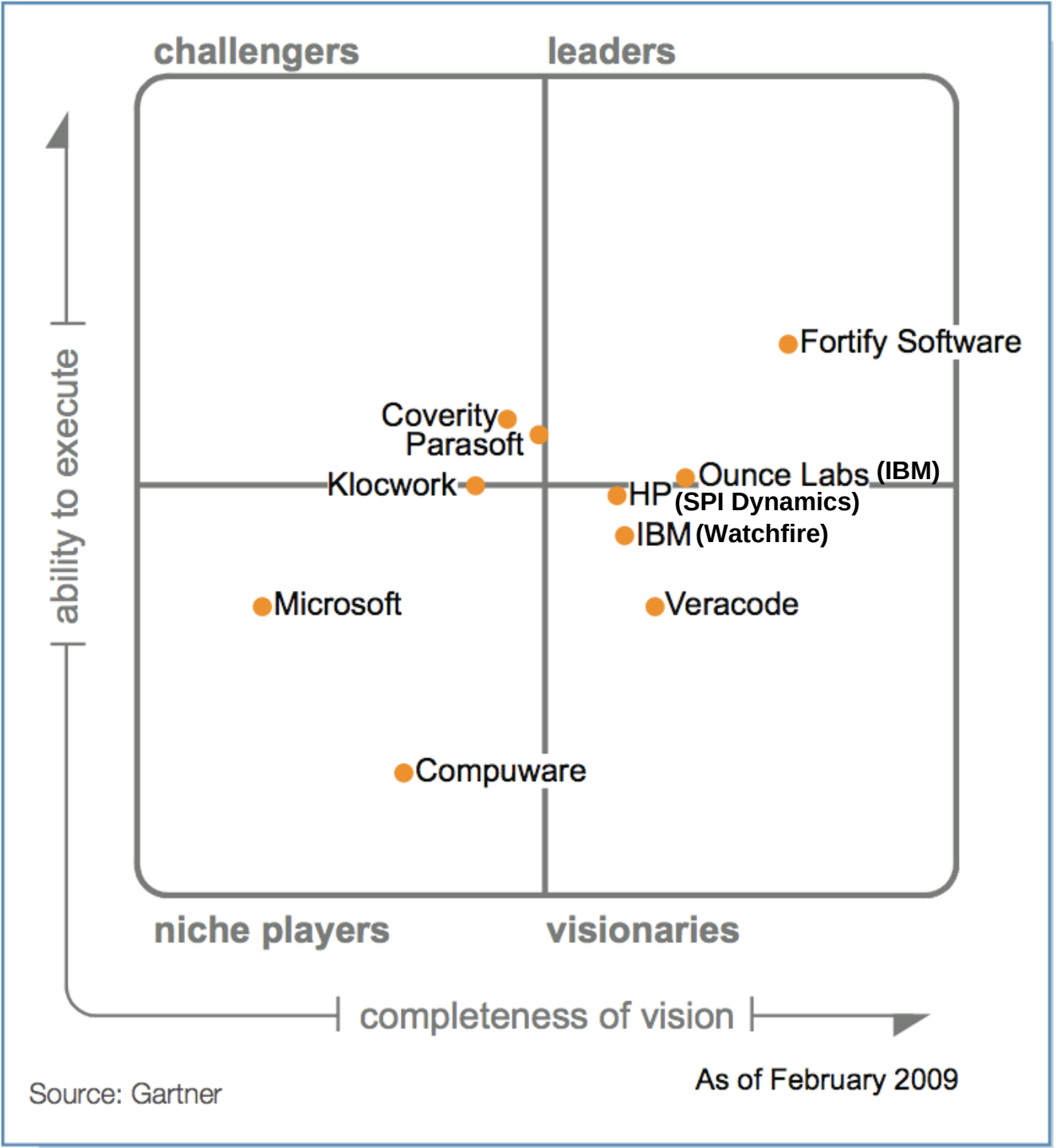


Modern Mission-Critical Software Systems are Multi-Platform, Multi-Language, and Multi-Sourced


Source Code
• Oracle PL/SQL
• Sybase T-SQL
• SQL Server T-SQL
• IBM SQL/PSM
• C
• C++
• Pro C
• Cobol
• CICS
• Visual Basic
• OracleForms
• Power Builder
• C#
• VB.Net
• ASPNet
• Java
• JSP
• XML
• HTML
• Javascript
• VBScript
• PHP
• SAP
• Tibco
• Business Objects
• Universal Analyzer for Other Languages
Knowledge Base
Oracle
Sybase
DB2
SQL Server
Engines
Quality Engine
Compliance Engine
Structure Engine


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 dat
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



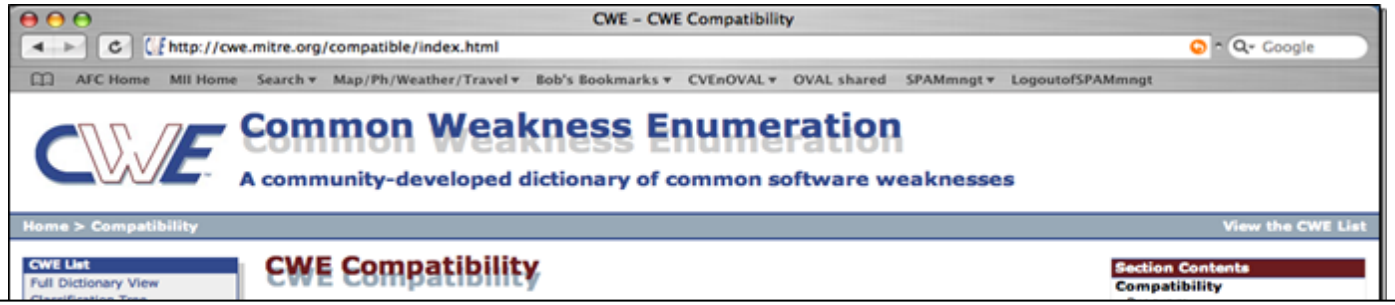
Gartner Magic Quadrant for Static Application Security Testing Tools

Plus Some Other Important Players…

Cenzic
proServices
Polyspace
Security Innovation
AppSIC Initiative
KDManalytics
SureLogic
Programming Research Inc
Armorize
Compuware
SofCheck
GrammaTech
CORE Security

CWE Compatibility & Effectiveness Program

(launched Feb 2007)



SECURITY DATABASE
KEEP A VIGILANT EYE ON YOUR DEFENSES
KDM Analytics
VERACODE
FORTIFY SOFTWARE
armorize technologies
Secure Your Web Code
SecurityReason
Our Reason is Security
SkillBridge
coverity
Klocwork
PARASOFT
EMC²
CODENOMICON defensics
CENZIC
GRAMMATECH
CERIAS
Center for Education and Research in Information Assurance and Security
IPA
INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN
SofCheck
IBM
watchfire
checkmarx
SANS
(ISC)²
OUNCE LABS
PRQA
Programming Research
CAST
LDRA
Software Technology
cigital
hp
SPI DYNAMICS

Organizations Participating

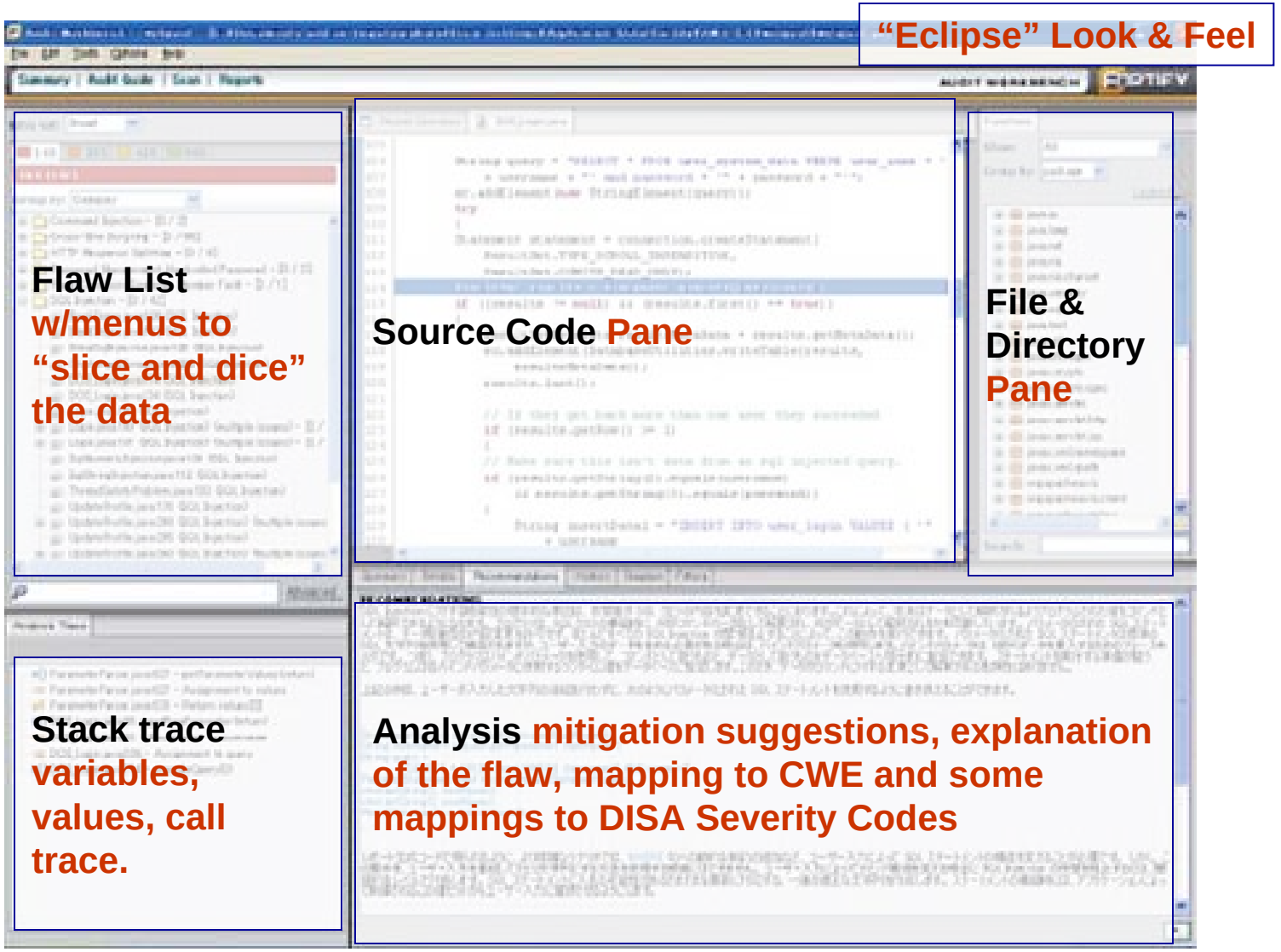
All organizations participating in the CWE Compatibility and Effectiveness Program are listed below, including those with CWE-Compatible Products and Services and those with Declarations to Be CWE-Compatible.

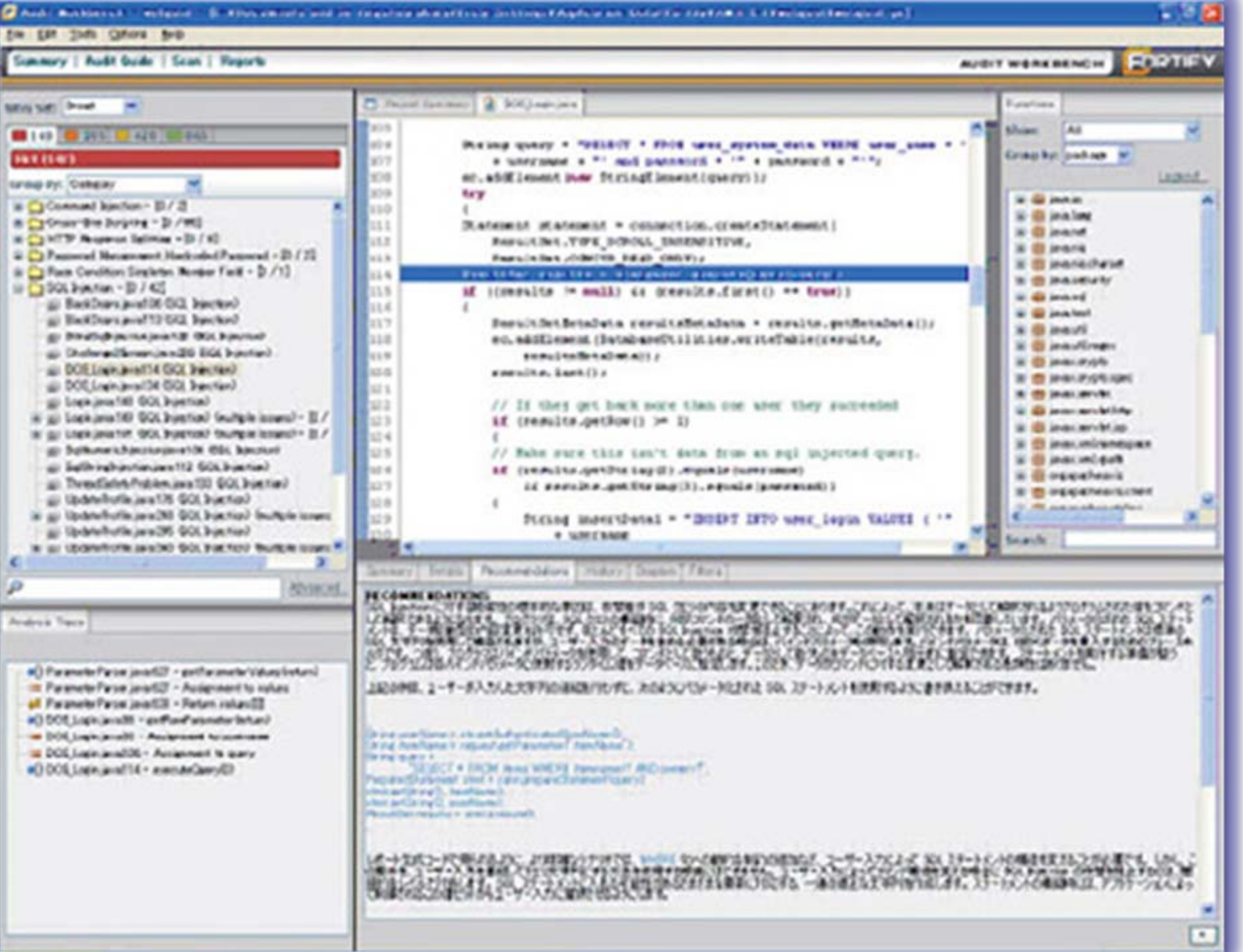
TOTALS
Organizations Participating: **28**
Products & Services: **47**

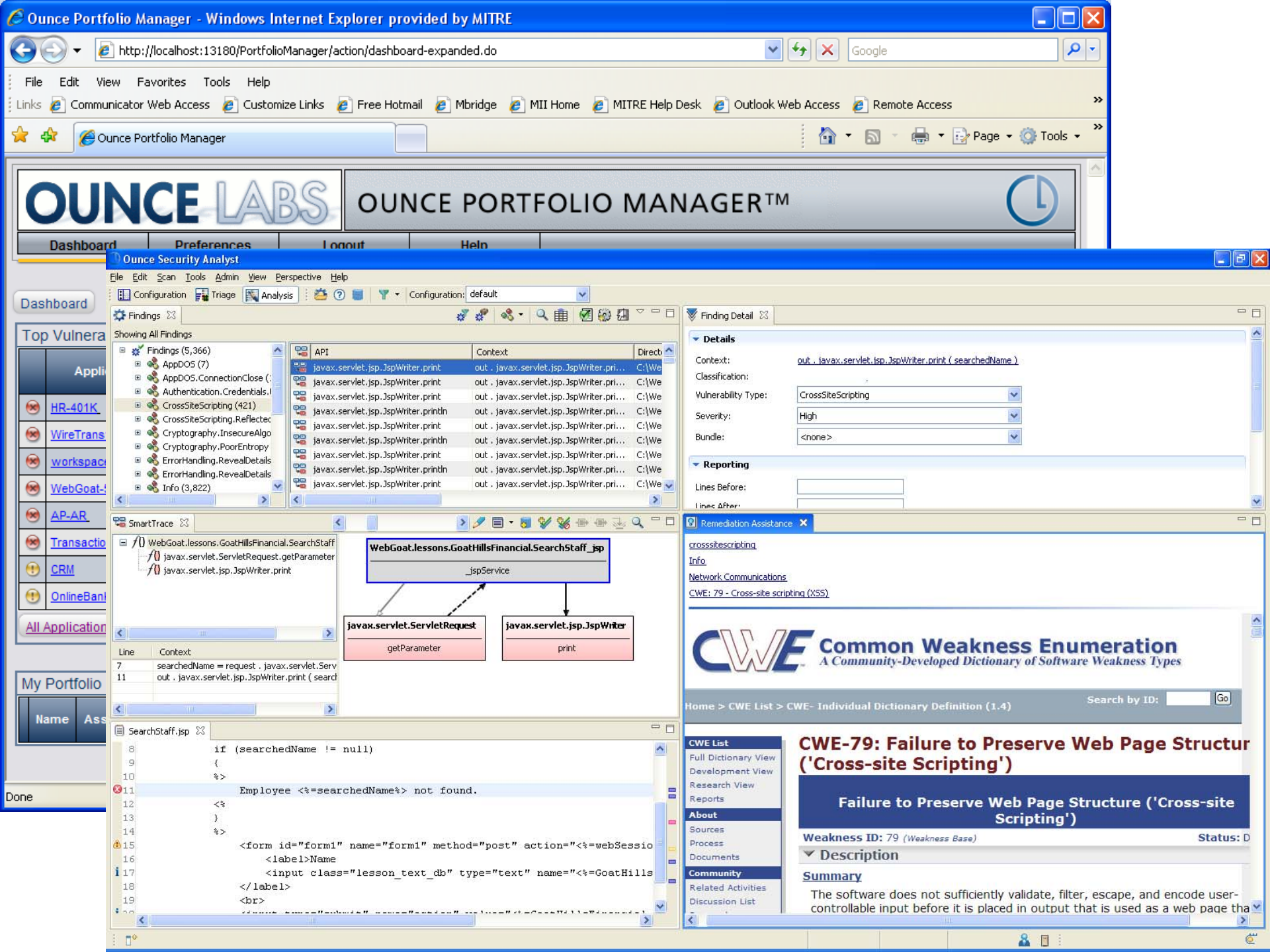
Products are listed alphabetically by organization name:

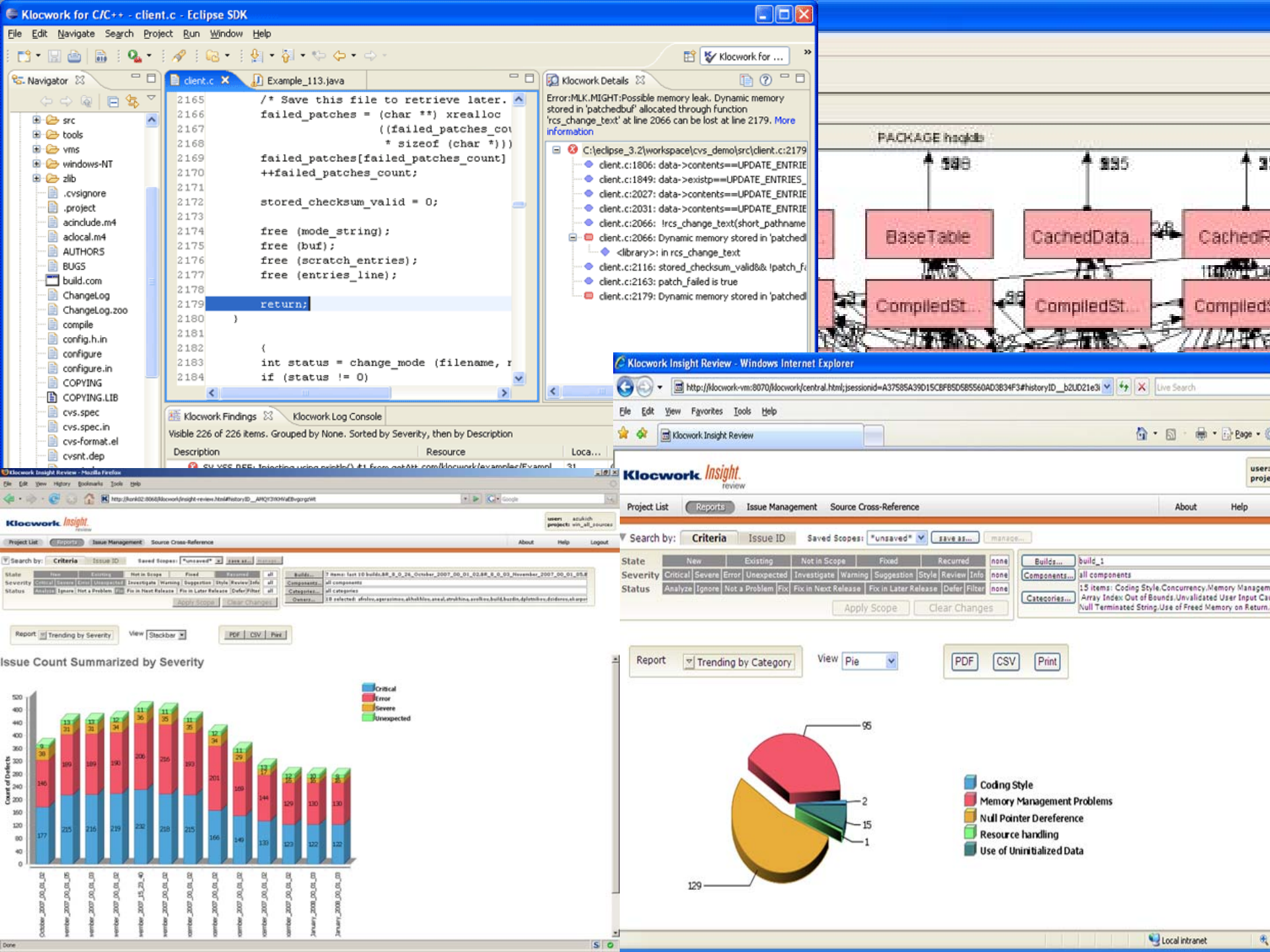
cwe.mitre.org/compatible/

Fortify Main User Interface

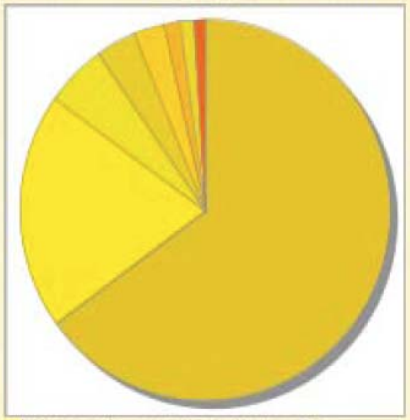



Summary | Audit Guide | Scan | Reports
AUDIT WORKBENCH
FORTIFY
Group by: Category
Recommendations


Ounce Portfolio Manager - Windows Internet Explorer provided by MITRE
http://localhost:13180/PortfolioManager/action/dashboard-expanded.do
File Edit View Favorites Tools Help
Ounce Portfolio Manager
OUNCE LABS
OUNCE PORTFOLIO MANAGER™
Dashboard
Preferences
Logout
Help
Ounce Security Analyst
File Edit Scan Tools Admin View Perspective Help
Configuration Triage Analysis
Findings
Showing All Findings
Finding Detail
Details
Reporting
SmartTrace
WebGoat.lessons.GoatHillsFinancial.SearchStaff_jsp
_jspService
javax.servlet.ServletRequest
getParameter
javax.servlet.jsp.JspWriter
print
Remediation Assistance
Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types
CWE-79: Failure to Preserve Web Page Structur ('Cross-site Scripting')
Failure to Preserve Web Page Structure ('Cross-site Scripting')
Description
Summary
SearchStaff.jsp

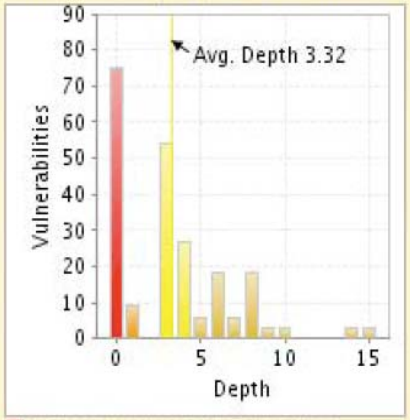

Klocwork for C/C++ - client.c - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Navigator
client.c
Example_113.java
Klocwork Details
Error:MLK.MIGHT:Possible memory leak. Dynamic memory stored in 'patchedbuf' allocated through function 'rcs_change_text' at line 2066 can be lost at line 2179. More information
Klocwork Findings
Klocwork Log Console
Visible 226 of 226 items. Grouped by None. Sorted by Severity, then by Description
Description
Resource
PACKAGE hsqldb
BaseTable
CachedData...
CompiledSt...
Klocwork Insight Review - Windows Internet Explorer
Klocwork Insight
review
Project List
Reports
Issue Management
Source Cross-Reference
About
Help
Search by:
Criteria
Issue ID
Saved Scopes:
Apply Scope
Clear Changes
Issue Count Summarized by Severity
Critical
Error
Severe
Unexpected
Report
Trending by Category
View
Pie
PDF
CSV
Print
Coding Style
Memory Management Problems
Null Pointer Dereference
Resource handling
Use of Uninitialized Data
95
2
15
1
129

Vulnerability Type Distribution



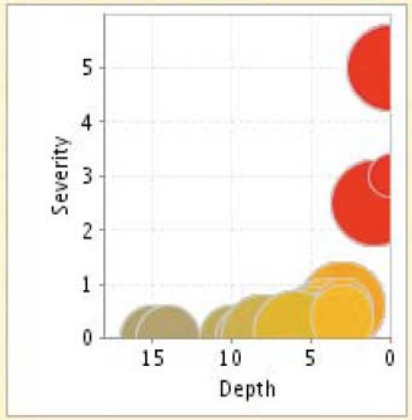
- Cross-Site Scripting (CWE 79)
- SQL Injection (CWE 89)
- Resource Injection (CWE 99)
- HTTP Response Splitting (CWE 113)
- Command Injection (CWE 77)
- Hard-Coded Password (CWE 259)
- XPath Injection (CWE 91)
- Information Leak of System Data (CWE 497)

Vulnerability Depth Distribution



- Directly expo
- Vulnerabilitie
- Vulnerabilitie
- Vulnerabilitie

Vulnerability Severity Distribution



Web Application Secu

Multi-Language Support

J2EE (Java, JSP)

.NET

ASP

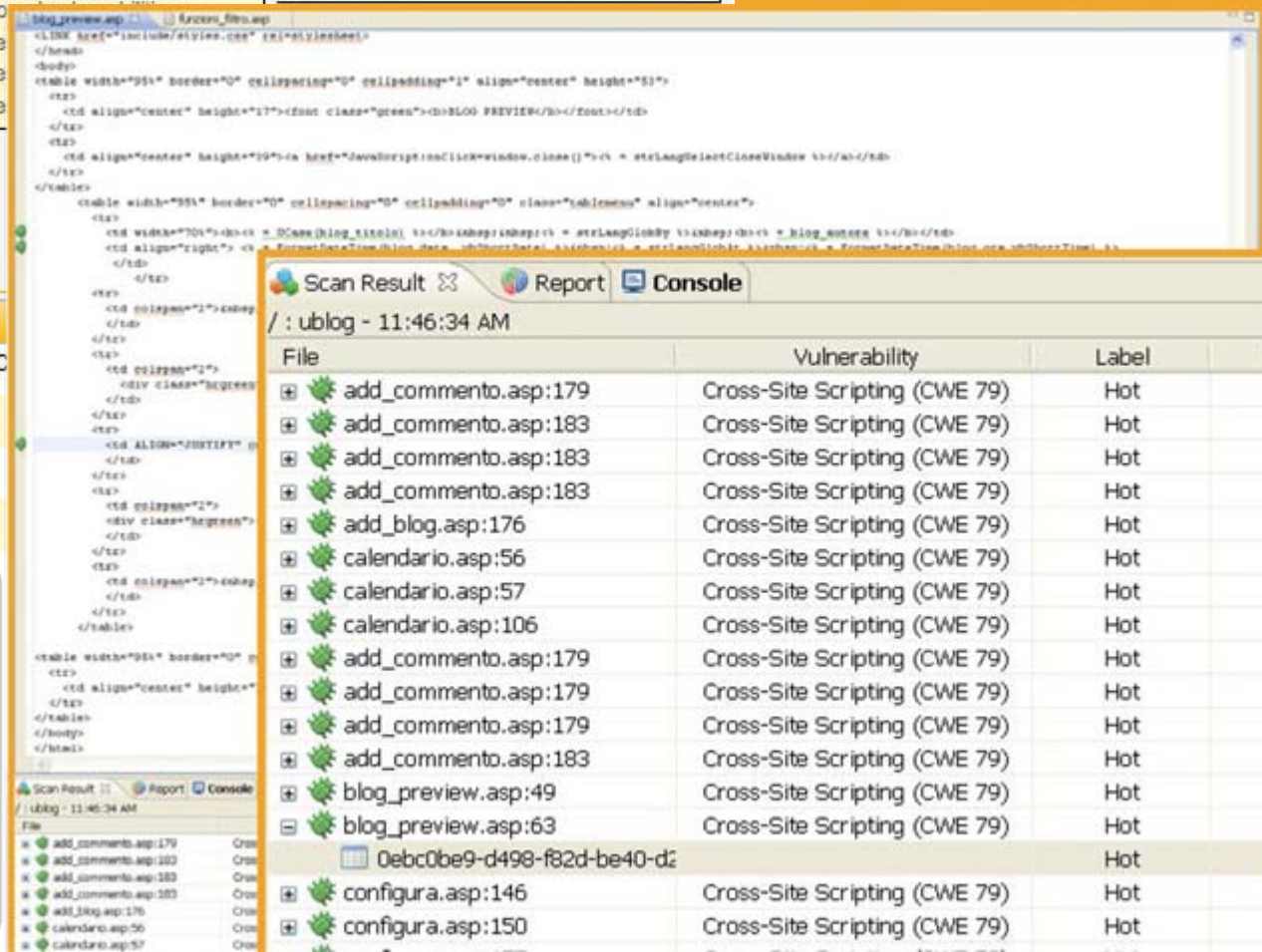
PHP

Vulnerability List

File Name	Vulnerability Type
DefaultLessonAction.java	Information Leak of System Data (C
EditProfile.jsp	Cross-Site Scripting (CWE 79)
EditProfile.jsp	Cross-Site Scripting (CWE 79)
EditProfile.jsp	Cross-Site Scripting (CWE 79)

Vulnerabilities Coverage

Cross-Site Scripting (CWE 79)
SQL Injection (CWE 89)
Command Injection (CWE 77)
File Inclusion (CWE 98)
Resource Injection (CWE 99)
Information Leak of System Data (CWE 497)
Hard-Coded Password (CWE 259)
Open Redirect (CWE 601)
XPath Injection (CWE 91)
API Abuse (CWE 227)
HTTP Response Splitting (CWE 113)
LDAP Injection (CWE 90)
Reflection Injection
Tag Injection



Scan Result | Report | Console

/ : ublog - 11:46:34 AM

File	Vulnerability	Label
add_commento.asp:179	Cross-Site Scripting (CWE 79)	Hot
add_commento.asp:183	Cross-Site Scripting (CWE 79)	Hot
add_commento.asp:183	Cross-Site Scripting (CWE 79)	Hot
add_commento.asp:183	Cross-Site Scripting (CWE 79)	Hot
add_blog.asp:176	Cross-Site Scripting (CWE 79)	Hot
calendario.asp:56	Cross-Site Scripting (CWE 79)	Hot
calendario.asp:57	Cross-Site Scripting (CWE 79)	Hot
calendario.asp:106	Cross-Site Scripting (CWE 79)	Hot
add_commento.asp:179	Cross-Site Scripting (CWE 79)	Hot
add_commento.asp:179	Cross-Site Scripting (CWE 79)	Hot
add_commento.asp:179	Cross-Site Scripting (CWE 79)	Hot
add_commento.asp:183	Cross-Site Scripting (CWE 79)	Hot
blog_preview.asp:49	Cross-Site Scripting (CWE 79)	Hot
blog_preview.asp:63	Cross-Site Scripting (CWE 79)	Hot
0ebc0be9-d498-f82d-be40-d2		Hot
configura.asp:146	Cross-Site Scripting (CWE 79)	Hot
configura.asp:150	Cross-Site Scripting (CWE 79)	Hot

Sample Warning

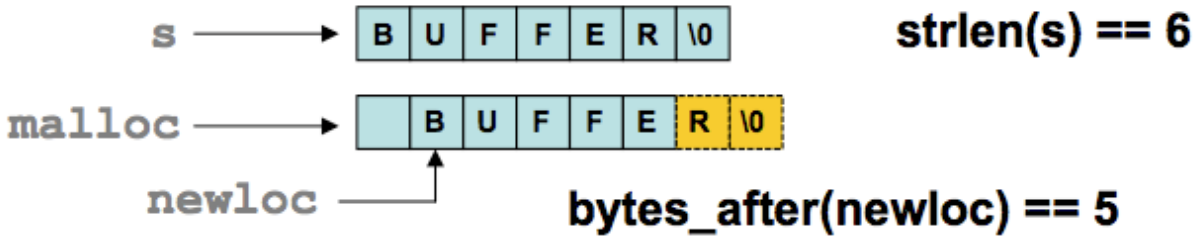
Buffer Overrun

Source

Problem	Line	Source
		C:\cygwin\home\Mark Zarins\codesonarexamples\gnuchess-5.07\src\lexpgn.c
		Enter return_append_str
	1766	`char *return_append_str(char *dest, const char *s) {`
	1767	`/* Append text s to dest, and return new result. */`
	1768	`char *newloc;`
	1769	`size_t newlen;`
	1770	`/* This doesn't have buffer overflow vulnerabilities, because`
	1771	`we always allocate for enough space before appending. */`
	1772	`if (!dest) {`
true	1773	`newloc = (char *) malloc(strlen(s))+1;`
strlen(s) > bytes_after(newloc) - 1	1774	`strcpy(newloc, s);  /* Buffer Overrun */`

+ Preconditions

+ Postconditions



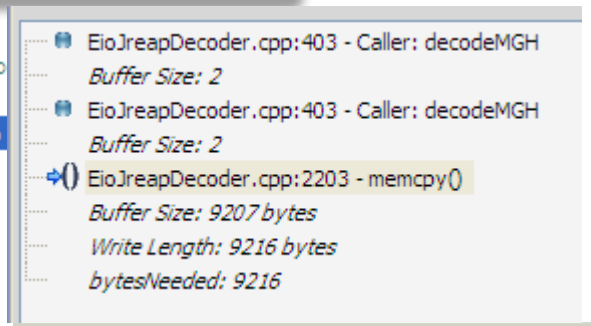
Example Buffer Overflow

Package:

File(Line): ExternalIoModel/ExternalIoPkg/…/**EioJreapDecoder.cpp:2203**

Function: memcpy(m_CrcBuffer, inputDB->getBufferWithOffset(), bytesNeeded);

```
2199          else
2200          {
2201              // Copy data straight into the crc buffer.  Calculation should b
2202              // any swap
2203              memcpy(m_CrcBuffer, inputDB->getBufferWithOffset(), bytesNeeded)
2204              offset = JREAP_MGH_CRC_WORD_OFFSET * JREAP_WORD_SIZE;
2205              // zero out the CRC to make the calculation
2206              m_CrcBuffer[offset] = 0;
2207              m_CrcBuffer[offset+1] = 0;
2208
2209              crcCalc = EioUtilities::calculateCRC(m_CrcBuffer, bytesNeeded);
2210
2211              // Offset is already be pointing at the CRC
2212              USHORT crcRcv = EioUtilities::create16BitInt( pBuff[offset+1], pB
2213              if(crcCalc != crcRcv)
2214              {
```



Buffer Overflow (Input Validation and Representation, buffer)

The program writes outside the bounds of allocated memory, which could corrupt data, crash the program, or lead to the execution of malicious code.

The **memcpy()** call copies **bytesNeeded** amount of data into **m_CrcBuffer**

- **m_CrcBuffer** is **MAX_CRC_BUFFER_SIZE**, which is 9207 bytes

- **bytesNeeded** = (**numJreapWords** * **NUM_BYTES_IN_JREAP_WORD**) + **NUM_BYTES_IN_HIGH**
- These are: (1024 * (72/8)) + 9

9216 bytes of input data is written into a buffer that can hold 9207 bytes **(overflow)**

Example Buffer Overflow: Off-by-One

Package:	CrbsModel->CrbsPkg->ConfigurationPkg->DataReductionRulesPkg
File:line:	src\CrbsModel\CrbsPkg\ConfigurationPkg**DataReductionRulesPkg:204**
Function:	few, including the obvious strcat(buffer, temp)

```
187 TRUEFALSE
188     //#[ op
189     CINT FI
190     CINT NUM_FIELDS = 4;
191     CUINT NUM_BITS = FIELD_SIZE * NUM_FIELDS;
192     CHAR buffer[NUM_BITS];
193     convertToBinary(codedValue, buffer);
194     UINT size = strlen(buffer);
195
196     if (size < NUM_BITS) {
197         CHAR temp[NUM_BITS];
198         memcpy(temp, buffer, size);
199         strcpy(buffer, "0");
200
201         while (strlen(buffer) + size < NUM_BITS)
202             strcat(buffer, "0");
203
204         strcat(buffer, temp);
205     }
```

-by-One (Input Validation
sentation, buffer)

The function reduceIcaoCode() in CrbsSpecialReduction.cpp writes one location past the bounds of buffer on line 204, which could corrupt data, cause the program to crash, or lead to the execution of malicious code.

CrbsSpecialReduction.cpp:192 - Buffer buffer Declared
CrbsSpecialReduction.cpp:204 - strcat()
Buffer Size: 20 bytes
Write Length: 21 bytes

a. "Magic numbers" declared locally & buffer declared on the stack as 5 * 4 (= 20) char bytes

b. Buffer is filled with values via convertToBinary call **(overflow potential?)**

c. Performance expensive code → duplicates **sprintf()** call poorly

   1. Duplicates the buffer into a temp buffer
   2. Overwrites the original buffer with an expensive **strcpy()** call → duplicates simple assignments
   3. Repeatedly recalculates the length of the buffer, while appending a '0' character → **strlen()** expensive
   4. Blindly appends the original content to the end **(overflow)** → appends original content

Example Buffer Overflow: Signed

Package: ExternalIOPkg->EioJreapPkg

File(Line): ExternalIoModel/…/EioJreapPkg/**EioJreapController.cpp:1358**

Function: memcpy(m_OwnUnitIcmBufferData, getMbufP().getMbufPtr(), m_OwnUnitIcmSize);

```
1343
1344VOID EioJr
1345{
1346    INT queueIndex;
1347
1348    getSearchCriteriaR().clearAllAttr();
1349    getSearchCriteriaR().setCriteriaVal(EIO_CRITERIA_ICM_TYPE, ICM_TYPE_MCP_LINK
1350    queueIndex = m_FwdRcvMgr->searchQueue(&getSearchCriteriaR(), EioFwdRcvMgr::(
1351    if(queueIndex >= 0)
1352    {
1353       m_FwdRcvMgr->receive mIndex(getMbufP(), queueIndex, ioFwdRcvMgr::QU
1354       m_OwnUnitLinkDataSize = getMbufP().getCriteria(EIO_CRITERIA_LINKDATA_S
1355       m_OwnUnitIcmSize      = getMbufP().getCriteria(EIO_CRITERIA_ICM_SIZE);
1356       if(m_OwnUnitIcmSize <= EIO_JREAP_PPLI_DATA_SIZE)
1357       {
1358          memcpy(m_OwnUnitIcmBufferData, getMbufP().getMbufPtr(), m_OwnUnitIc
1359          m_OwnUnitBufferValid = true;
```

Buffer Overflow: Signed Comparison (Input Validation and Representation, buffer)

The program uses a signed comparison to check a value that is later treated as unsigned. This could lead the program to write outside the bounds of allocated memory, which could corrupt data, crash the program, or lead to the execution of malicious code.

EioJreapController.cpp:1356 - Signed Comparison
EioJreapController.cpp:1358 - memcpy()
Buffer Size: 368 bytes
Write Length: (very large value) bytes

a. The m_OwnUnitIcmSize value is looked up via getMbufP().getCtiteria(EIO_CRITERIA_ICM_SIZE)

b. If the (unsigned integer) m_OwnUnitIcmSize is less than, or equal to (integer), EIO_JREAP_PPLI_DATA_SIZE

c. Then copy that m_OwnUnitIcmSize number of bytes from the pointer returned by the getMbufP().getMbufPtr() call into the m_OwnUnitIcmBufferData memory location.

d. Large, positive integer values will flip the highest bit. A signed comparison would consider that value to be negative.

e. Extremely large values to slip past the protective “if” statement and on to memcpy() (**overflow**)

AD Governance Dashboard

CAST AD Governance Dashboard let you measure, monitor and control application risk factors by assessing their source code for quality risks and technical structure and size.
Application code quality assessment results are presented using color-coded status (Red/Unacceptable, Orange/To Justify, Yellow/Acceptable, and Green/Excellent) and a 1-to-4 decimal grade (the higher the score, the better).

Focus on Application Legacy HR application - part of HR System - for Apr '06 snapshot

Assessment of the selected component. Click on the hyperlink above to zoom out this component. Click on the -History- hyperlink to compare values on all available snapshots. History

Quality and Quantity

Assessment of the quality and quantity of the selected component.

Quality of Application Legacy HR application in Apr '06 snapshot

Assessment of the quality of the selected component. Click on the hyperlinks below to see the application risk factors for the current context - component and snapshot -.

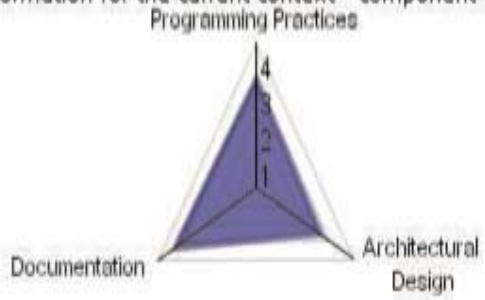
Transferability	3.03
Changeability	3.17
Robustness	3.13
Performance	3.8
Security	3.25

Maintainability (SEI)	3.43	3.43

SEI Maintainability assesses the cost and difficulty/ease to maintain an application in the future.

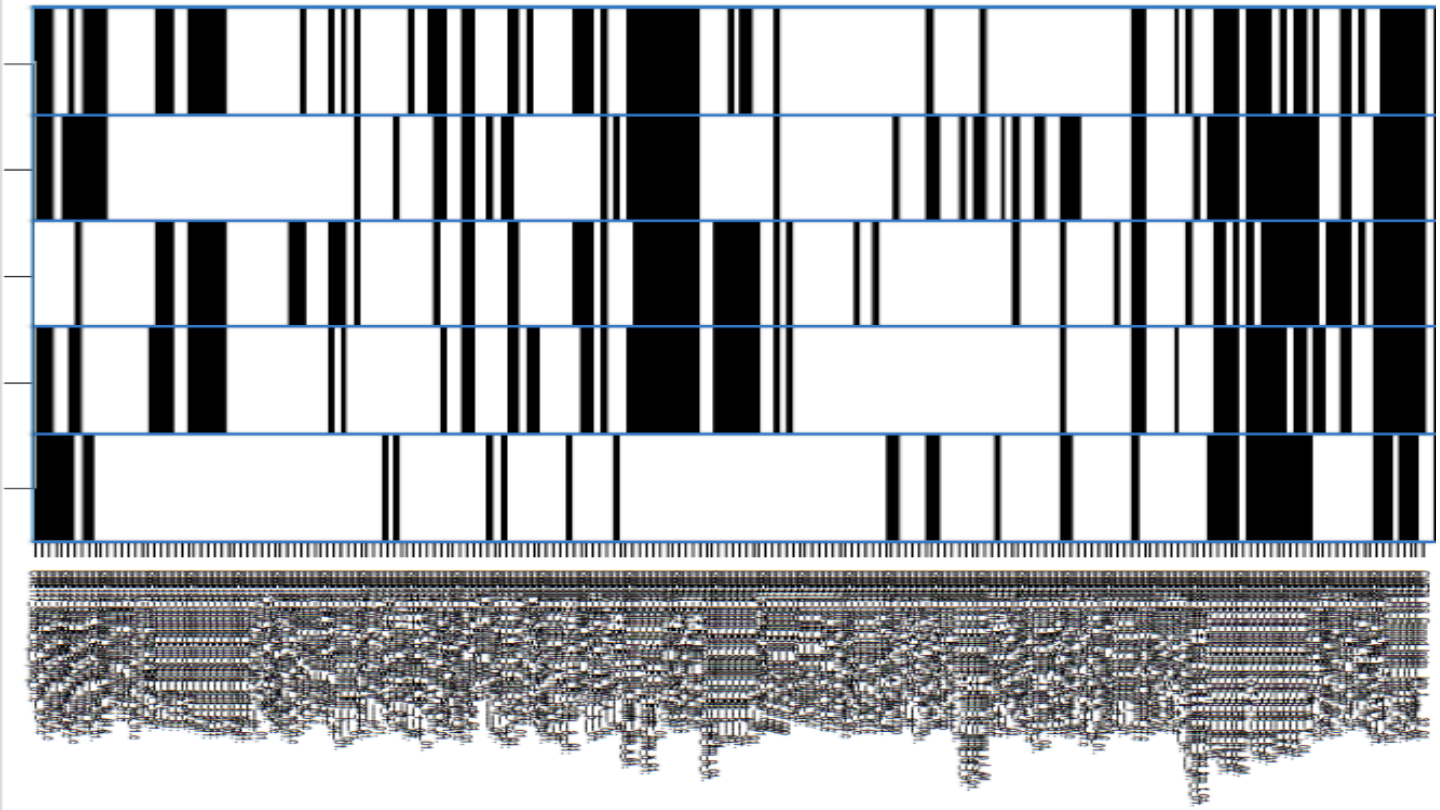
Rule Compliance of Application Legacy HR application in Apr '06 snapshot

Assessment of the compliance to rules for the selected component. Click on the hyperlinks below to drilldown on rule compliance information for the current context - component and snapshot -.

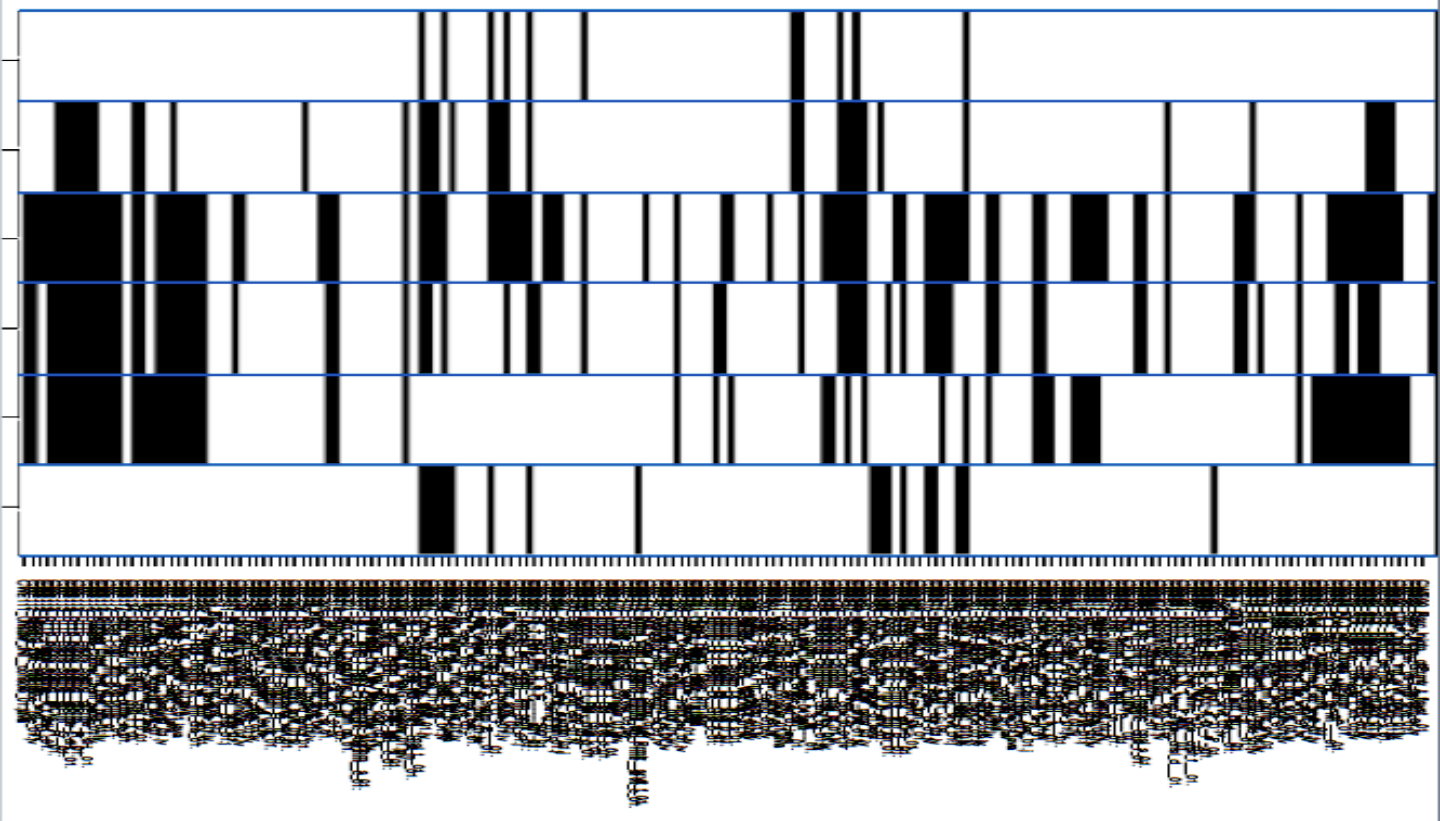


Programming Practices	2.97
Architectural Design	2.6
Documentation	3.34

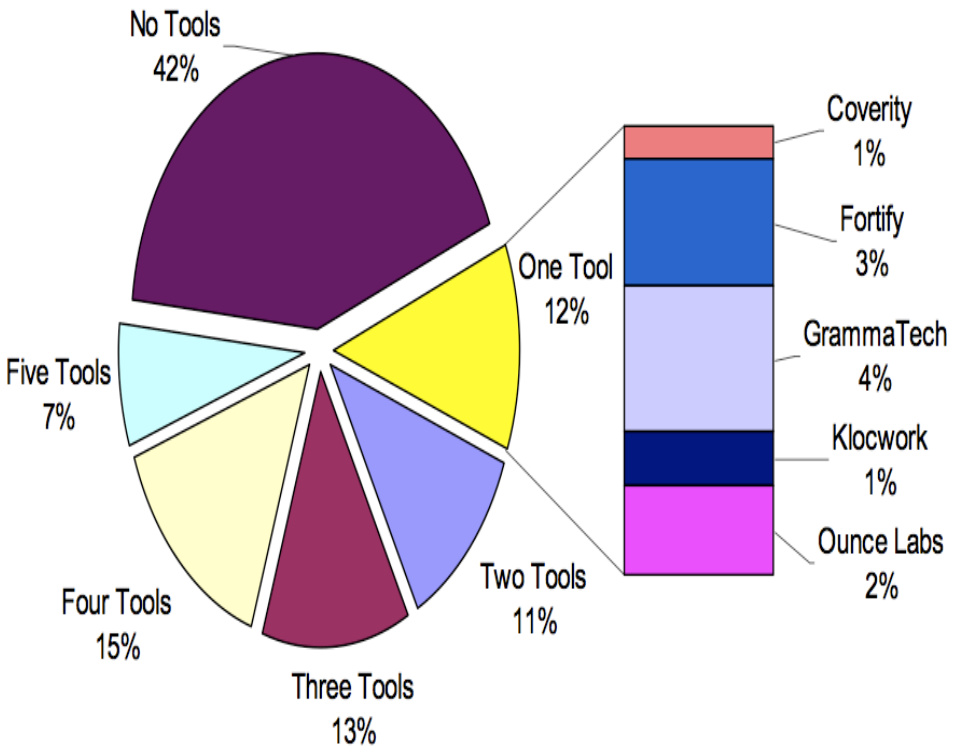
C Test Cases



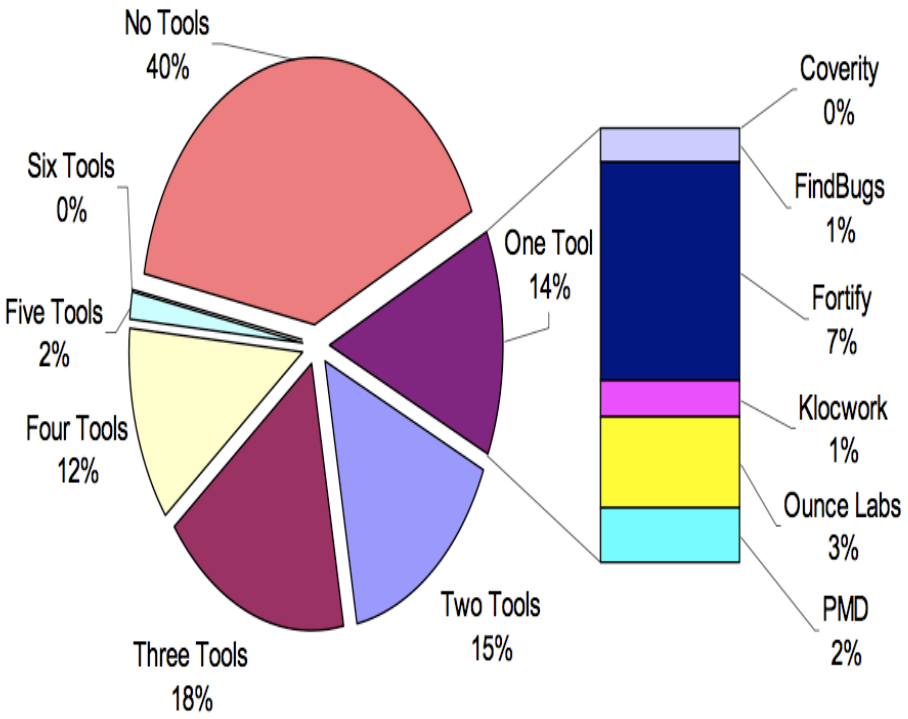
Java Test Cases



C/C++ "Breadth" Test Case Coverage



Java "Breadth" Test Case Coverage



Individual CAPEC Dictionary Definition (Release 1.2)

Blind SQL Injection

Attack Pattern ID 7

Pattern Abstraction: Detailed

Typical Severity High

Description

Summary

Blind SQL Injection results from an insufficient mitigation for SQL Injection. Although suppressing database error messages are considered best practice, the suppression alone is not sufficient to prevent SQL Injection. Blind SQL Injection is a form of SQL Injection that overcomes the lack of error messages. Without the error messages that facilitate SQL Injection, the attacker constructs input strings that probe the target through simple Boolean SQL expressions. The attacker can determine if the syntax and structure of the injection was successful based on whether the query was executed or not. Applied iteratively, the attacker determines how and where the target is vulnerable to SQL Injection.

In order to achieve this using Blind SQL Injection, an attacker:

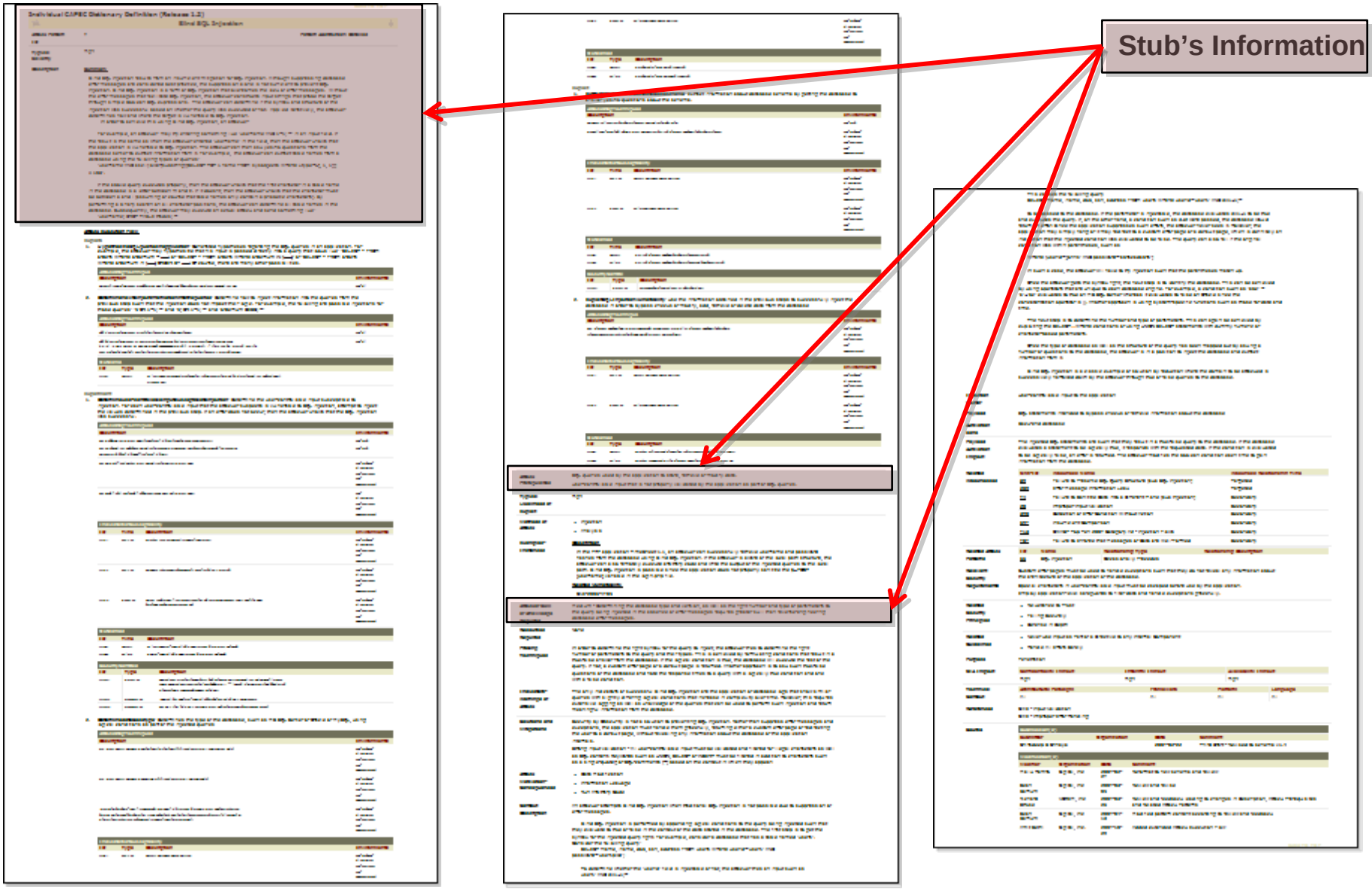
For example, an attacker may try entering something like "username' AND 1=1; --" in an input field. If the result is the same as when the attacker entered "username" in the field, then the attacker knows that the application is vulnerable to SQL Injection. The attacker can then ask yes/no questions from the database server to extract information from it. For example, the attacker can extract table names from a database using the following types of queries:

"username' AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERE xtype='U'), 1, 1))) > 108".

If the above query executes properly, then the attacker knows that the first character in a table name in the database is a letter between m and z. If it doesn't, then the attacker knows that the character must be between a and l (assuming of course that table names only contain alphabetic characters). By performing a binary search on all character positions, the attacker can determine all table names in the database. Subsequently, the attacker may execute an actual attack and send something like:

"username'; DROP TABLE trades; --

Complete CAPEC Entry Information



CAPEC Current Content (12 Major Categories)

1000 - Mechanism of Attack

Data Leakage Attacks - (118)

Resource Depletion - (119)

Injection (Injecting Control Plane content through the Data Plane) - (152)

Spoofing - (156)

Time and State Attacks - (172)

Abuse of Functionality - (210)

Probabilistic Techniques - (223)

Exploitation of Authentication - (225)

Exploitation of Privilege/Trust - (232)

Data Structure Attacks - (255)

Resource Manipulation - (262)

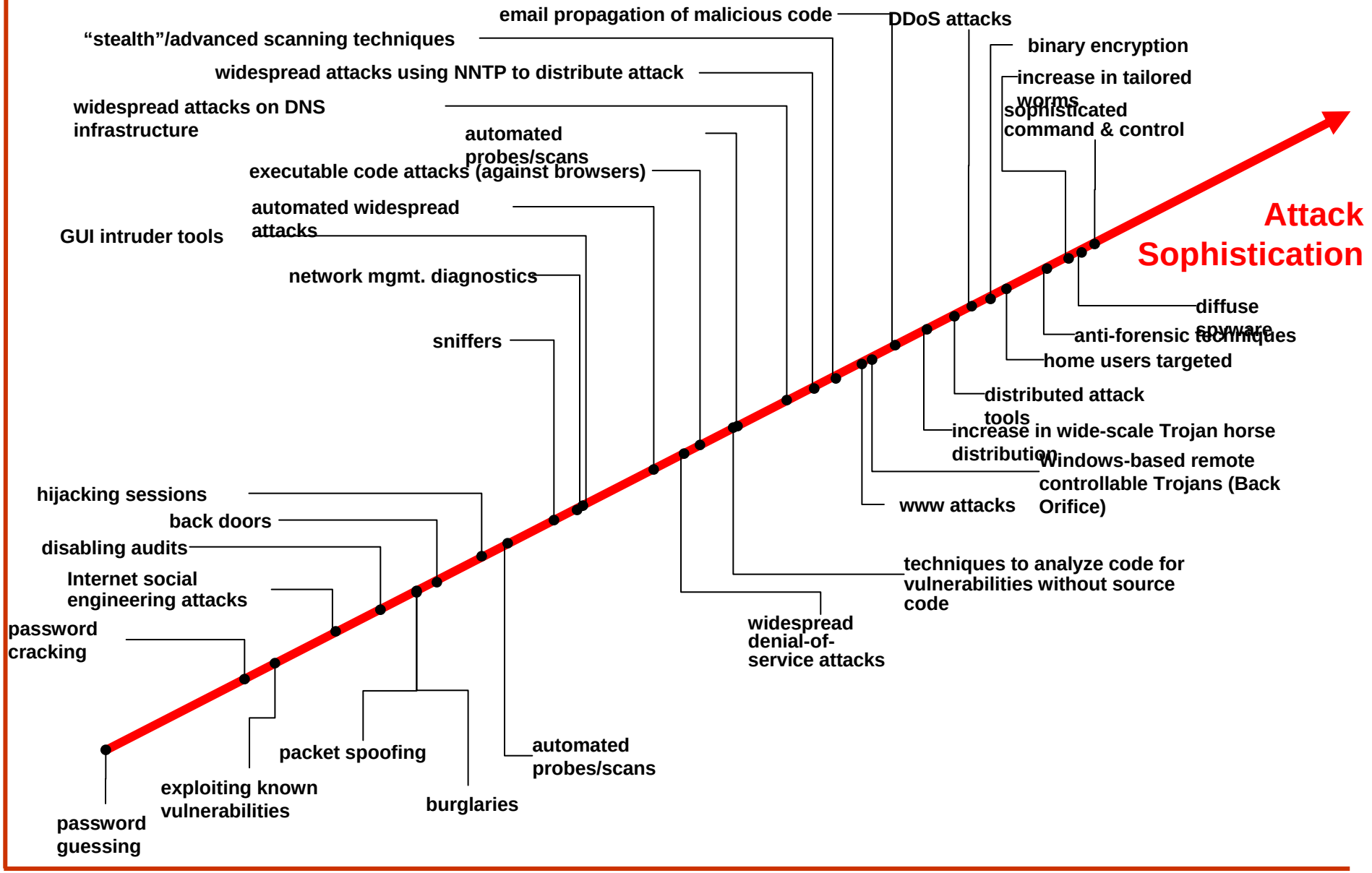
Network Reconnaissance - *(286)*

CAPEC Current Content (Which Expand to…)

- 1000 - Mechanism of Attack
  - Data Leakage Attacks - (118)
    - Data Excavation Attacks - (116)
    - Data Interception Attacks - (117)
  - Resource Depletion - (119)
    - Violating Implicit Assumptions Regarding XML Content (aka XML Denial of Service (XDoS)) - (82)
    - Resource Depletion through Flooding - (125)
    - Resource Depletion through Allocation - (130)
    - Resource Depletion through Leak - (131)
    - Denial of Service through Resource Depletion - (227)
  - Injection (Injecting Control Plane content through the Data Plane) - (152)
    - Remote Code Inclusion - (253)
    - Analog In-band Switching Signals (aka Blue Boxing) - (5)
    - SQL Injection - (66)
    - Email Injection - (134)
    - Format String Injection - (135)
    - LDAP Injection - (136)
    - Parameter Injection - (137)
    - Reflection Injection - (138)
    - Code Inclusion - (175)
    - Resource Injection - (240)
    - Script Injection - (242)
    - Command Injection - (248)
    - Character Injection - (249)
    - XML Injection - (250)
    - DTD Injection in a SOAP Message - (254)
  - Spoofing - (156)
    - Content Spoofing - (148)
    - Identity Spoofing (Impersonation) - (151)
    - Action Spoofing - (173)
  - Time and State Attacks - (172)
    - Forced Deadlock - (25)
    - Leveraging Race Conditions - (26)
    - Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions - (29)
    - Manipulating User State - (74)
  - Abuse of Functionality - (210)
    - Functionality Misuse - (212)
    - Abuse of Communication Channels - (216)
    - Forceful Browsing - (87)
    - Passing Local Filenames to Functions That Expect a URL - (48)
    - Probing an Application Through Targeting its Error Reporting - (54)
    - WSDL Scanning - (95)
    - API Abuse/Misuse - (113)
    - Try All Common Application Switches and Options - (133)
    - Cache Poisoning - (141)
    - Software Integrity Attacks - (184)
    - Directory Traversal - (213)
    - Analytic Attacks - (281)
  - Probabilistic Techniques - (223)
    - Fuzzing - (28)
    - Manipulating Opaque Client-based Data Tokens - (39)
    - Brute Force - (112)
    - Screen Temporary Files for Sensitive Information - (155)
  - Exploitation of Authentication - (225)
    - Exploitation of Session Variables, Resource IDs and other Trusted Credentials - (21)
    - Authentication Abuse - (114)
    - Authentication Bypass - (115)
  - Exploitation of Privilege/Trust - (232)
    - Privilege Escalation - (233)
    - Exploiting Trust in Client (aka Make the Client Invisible) - (22)
    - Hijacking a Privileged Thread of Execution - (30)
    - Subvert Code-signing Facilities - (68)
    - Target Programs with Elevated Privileges - (69)
    - Exploitation of Authorization - (122)
    - Hijacking a privileged process - (234)
  - Data Structure Attacks - (255)
    - Accessing/Intercepting/Modifying HTTP Cookies - (31)
    - Buffer Attacks - (123)
    - Attack through Shared Data - (124)
    - Integer Attacks - (128)
    - Pointer Attack - (129)
  - Resource Manipulation - (262)
    - Accessing/Intercepting/Modifying HTTP Cookies - (31)
    - Input Data Manipulation - (153)
    - Resource Location Attacks - (154)
    - Infrastructure Manipulation - (161)
    - File Manipulation - (165)
    - Variable Manipulation - (171)
    - Configuration/Environment manipulation - (176)
    - Abuse of transaction data strutcture - (257)
    - Registry Manipulation - (269)
    - Schema Poisoning - (271)
    - Protocol Manipulation - (272)
  - Network Reconnaissance - (286)
    - ICMP Echo Request Ping - (285)
    - TCP SYN Scan - (287)
    - ICMP Echo Request Ping - (288)
    - Infrastructure-based footprinting - (289)
    - Enumerate Mail Exchange (MX) Records - (290)
    - DNS Zone Transfers - (291)
    - Host Discovery - (292)
    - Traceroute Route Enumeration - (293)
    - ICMP Address Mask Request - (294)
    - ICMP Timestamp Request - (295)
    - ICMP Information Request - (296)
    - TCP ACK Ping - (297)
    - UDP Ping - (298)
    - TCP SYN Ping - (299)
    - Port Scanning - (300)
    - TCP Connect Scan - (301)
    - TCP FIN scan - (302)
    - TCP Xmas Scan - (303)
    - TCP Null Scan - (304)
    - TCP ACK Scan - (305)
    - TCP Window Scan - (306)
    - TCP RPC Scan - (307)
    - UDP Scan - (308)

CAPEC Current Content (305 Attacks…)

Cyber Threats Emerged Over Time



Cyber Threats

Threat Landscape

- Complex software and hardware have many vulnerabilities

- Many critical components developed by foreigners

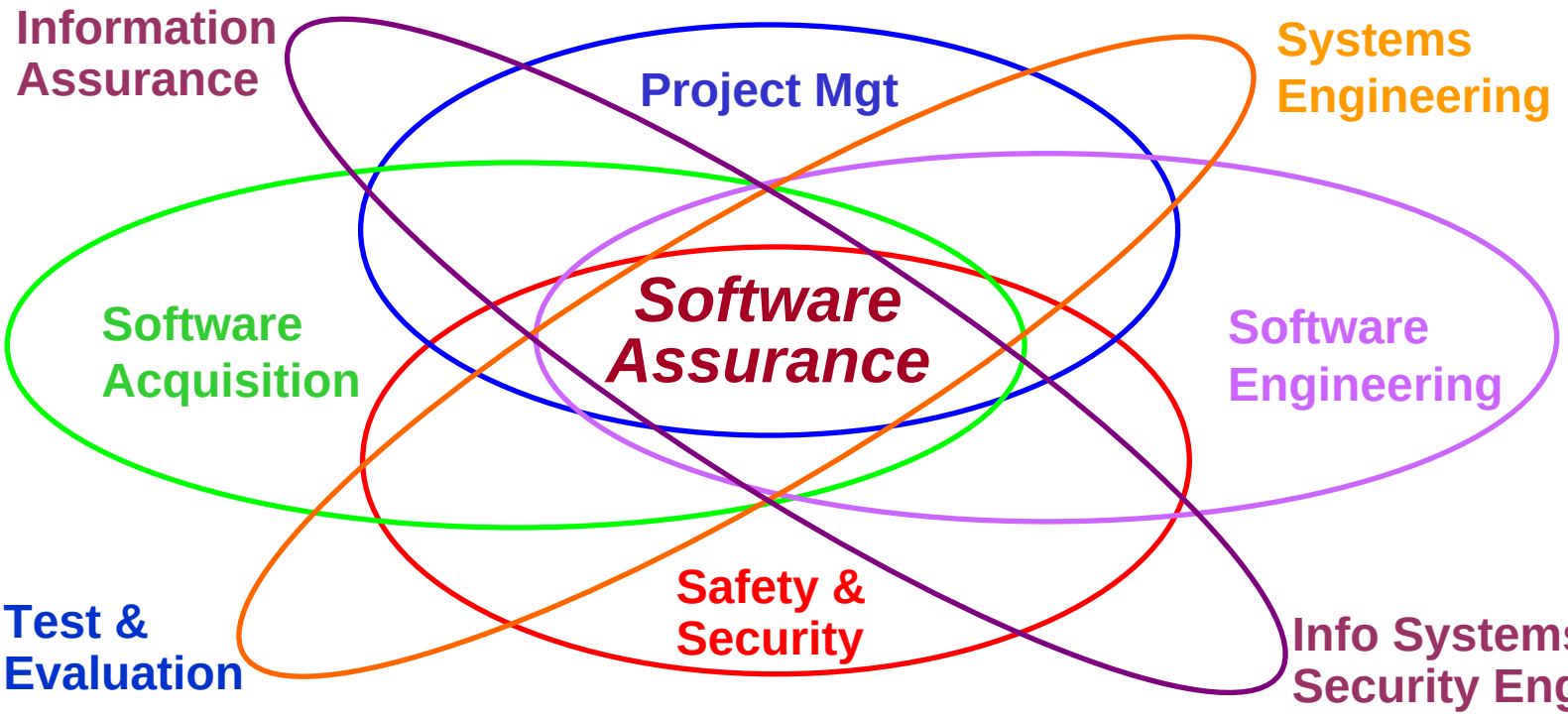
- IT supply
- Sophistic
  increasin

- "All" software "can" have vulnerabilities
- Critical software can have "un-vetted" creators
- Many places to "attack" the IT supply chain

But the threat to DoD systems is from here

IA in the DoD weapons system acquisition process (until recently) was focused on this threat

What is Software Assurance (SwA)?



Software Assurance is not a separate new discipline but rather it is an extension to each of the disciplines involved in a System's Development

“Software Assurance”

(from http://en.wikipedia.org/wiki/Software_Assurance)

Software Assurance (SwA) is: “the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle, and that the software functions in the intended manner”

— Source: Committee on National Security Systems (CNSS) Instruction No. 4009, “National Information Assurance Glossary”, Revised 2006 — http://www.cnss.gov/instructions.html

Alternate definitions:

[1] **Software Assurance (SwA)** addresses:

- **Trustworthiness - No exploitable vulnerabilities exist, either maliciously or unintentionally inserted;**
- **Predictable Execution - Justifiable confidence that software, when executed, functions as intended;**
- **Conformance - Planned and systematic set of multi-disciplinary activities that ensure software processes and products conform to requirements, standards/ procedures.**

- Source: Department of Homeland Security “Build Security In” web portal – https://buildsecurityin.us-cert.gov/portal

[2] **Software Assurance (SwA)** relates to "the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software."

- Source: DoD Software Assurance Initiative, 13 September 2005 - https://acc.dau.mil/CommunityBrowser.aspx?id=25749

[3] **Software Assurance (SwA)** – is “the planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures to help achieve:

- **Trustworthiness - No exploitable vulnerabilities exist, either malicious or unintentionally origin, and**
- **Predictable Execution - Justifiable confidence that software, when executed, functions as intended.**

- Source: National Institute for Standards and Technology (NIST) - http://samate.nist.gov

[4] **Software Assurance** - "Planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures. It includes the disciplines of Quality Assurance, Quality Engineering, Verification and Validation, Nonconformance Reporting and Corrective Action, Safety Assurance, and Security Assurance and their application during a software life cycle.“

- Source: NASA-STD-2201-93 "Software Assurance Standard", 10 November 1992 - http://satc.gsfc.nasa.gov/assure/astd.txt

[5] **Software Assurance (SwA)** is “justifiable trustworthiness in meeting established business and security objectives.”

- Source: Object Management Group (OMG) – http://adm.org/SoftwareAssurance.pdf and

DHS - Challenges in Software Assurance

- Software vulnerabilities jeopardize infrastructure operations, business operations & services, intellectual property, and consumer trust
- Adversaries have capabilities to subvert the software supply chain:
  - Lifecycle processes offer opportunities to insert malicious code and to poorly design and build software which enables future exploitation
  - Government and businesses rely on COTS products and commercial developers using foreign and non-vetted domestic suppliers to meet majority of system requirements
  - Off-shoring magnifies risks and creates new threats to security, business property and processes, and individuals' privacy – requires domestic strategies to mitigate those risks
- Growing concern about inadequacies of suppliers' capabilities to build/deliver secure software – too few practitioners with requisite knowledge and skills
  - Current education & training provides too few practitioners with requisite competencies in secure software engineering – enrollment down in critical software-related degree programs
  - Competition in higher-end skills is increasing – implications for individuals, companies, & countries
  - Concern about suppliers and practitioners not exercising "minimum level of responsible practice"
- Processes and technologies are required to build trust into software

Homeland Security

Strengthen operational resiliency

DoD Perspective on the Software Assurance (SwA) Problem

- Software is critical to the Global Information Grid, most weapons, business and support systems
- DoD Perspective
  - **Targeted attacks**
    - Attacks from Nation-state, terrorist, criminal, rogue developers
  - **Unique Assets - NSS/Weapons**
  - **Types of Attacks**
    - Intentionally implanted logic (e.g., back doors, logic bombs, spyware)
    - Unintentional vulnerabilities maliciously exploited (e.g., poor quality or fragile code)
  - **Ability to exploit vulnerabilities remotely**
- Through software, the enemy may
  - **Steal or alter mission critical data**
  - **Corrupt or deny the function of mission critical platforms**

DoD OASD - Software Assurance is Critical*

- Software is the core constituent of modern products and services – it enables functionality and business operations
- Dramatic increase in mission risk due to increasing:
  - **Software dependence and system interdependence (weakest link syndrome)**
  - **Software Size & Complexity (obscures intent and precludes exhaustive test)**
  - **Outsourcing and use of un-vetted software supply chain (COTS & custom)**
  - **Attack sophistication (easing exploitation)**
  - **Reuse (unintended consequences increasing number of vulnerable targets)**
  - **Number of vulnerabilities & incidents with threats targeting software**
  - **Risk of Asymmetric Attack and Threats**
- Increasing awareness and concern

Software and the processes for acquiring and developing software represent a material weakness

* **[Source: Interim Report on "Software Assurance: Mitigating Software Risks in the DoD IT and National Security Systems," DoD OASD(NII) forwarded to Committee on National Security Systems (CNSS)), Oct 2004]**

Summary of the SwA Problem

- Systems are at risk due to software content & threat environment
- Software assurance is a significant part of Mission Assurance
- Significant risks come from
  - **Human coding mistakes and design flaws leading to security flaws**
  - **Supply Chain compromises**
- It is best to identify/avoid software flaws earlier
  - **But projects need the target list of weaknesses in code and activities as well as assurance methodologies for confirming that risks were adequately addressed**

Software Assurance's Challenges

- Software Assurance
  - Advanced capabilities to test and evaluate IT products
  - Identify IA standards and best practices
- Supply Chain Assurance
  - Develop "defense-in-breadth" policies and capabilities
  - Create national clearinghouse to collect, share threa information about IT suppliers

Find weaknesses in:

- software architecture,
- software design, and
- software implementation

that can lead to exploitable vulnerabilities in operations

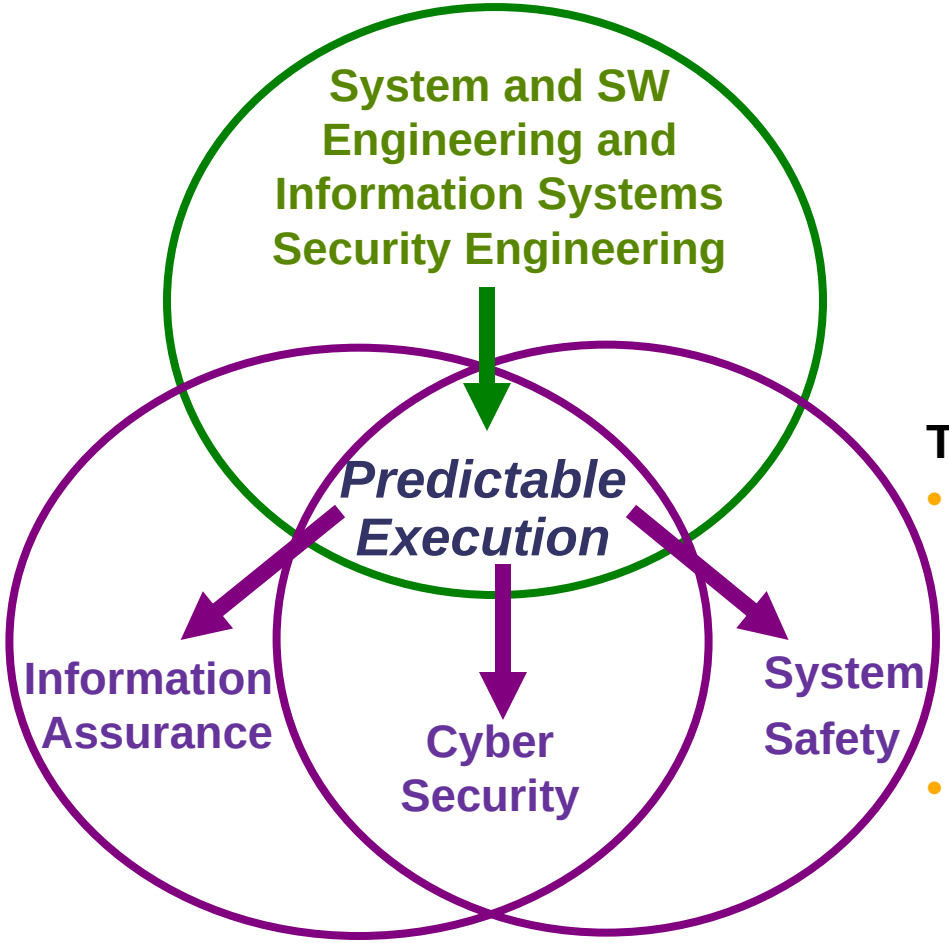
Need to address software:

- developed under contract
- purchased, and/or
- integrated libraries/modules

Understand all of the places software is injected into a system's supply chain and all of the technologies and organizations that can influence those software elements

The supply chain of interest is that which impacts software elements that end up in the final system and the system's sustainment capabilities

SwA's Relationship to Traditional System/Software Engineering Disciplines



For a safety/security analysis to be valid …

The execution of the system must be *predictable*.

This requires …

- **Correct implementation of requirements, expectations and regulations.** — ***Traditional concern***
- **Exclusion of unwanted function even in the face of attempted exploitation.** — ***Growing concern***

Predictable Execution = requisite enabling characteristic

***Adopted from Jim Moore, IEEE CS S2ESC Liaison to ISO SC7**

“Software Assurance” Comes From:

Knowing what it takes to “get” what we want

- **Development/acquisition practices/process capabilities**
- **Criteria for assuring integrity & mitigating risks**

Building and/or acquiring what we want

- Threat modeling and analysis
- Requirements engineering
- Failsafe design and defect-free code
- Supply Chain Management

Understanding what we built / acquired

- **Production assurance evidence**
- **Comprehensive testing and diagnostics**
- **Formal methods & static analysis**

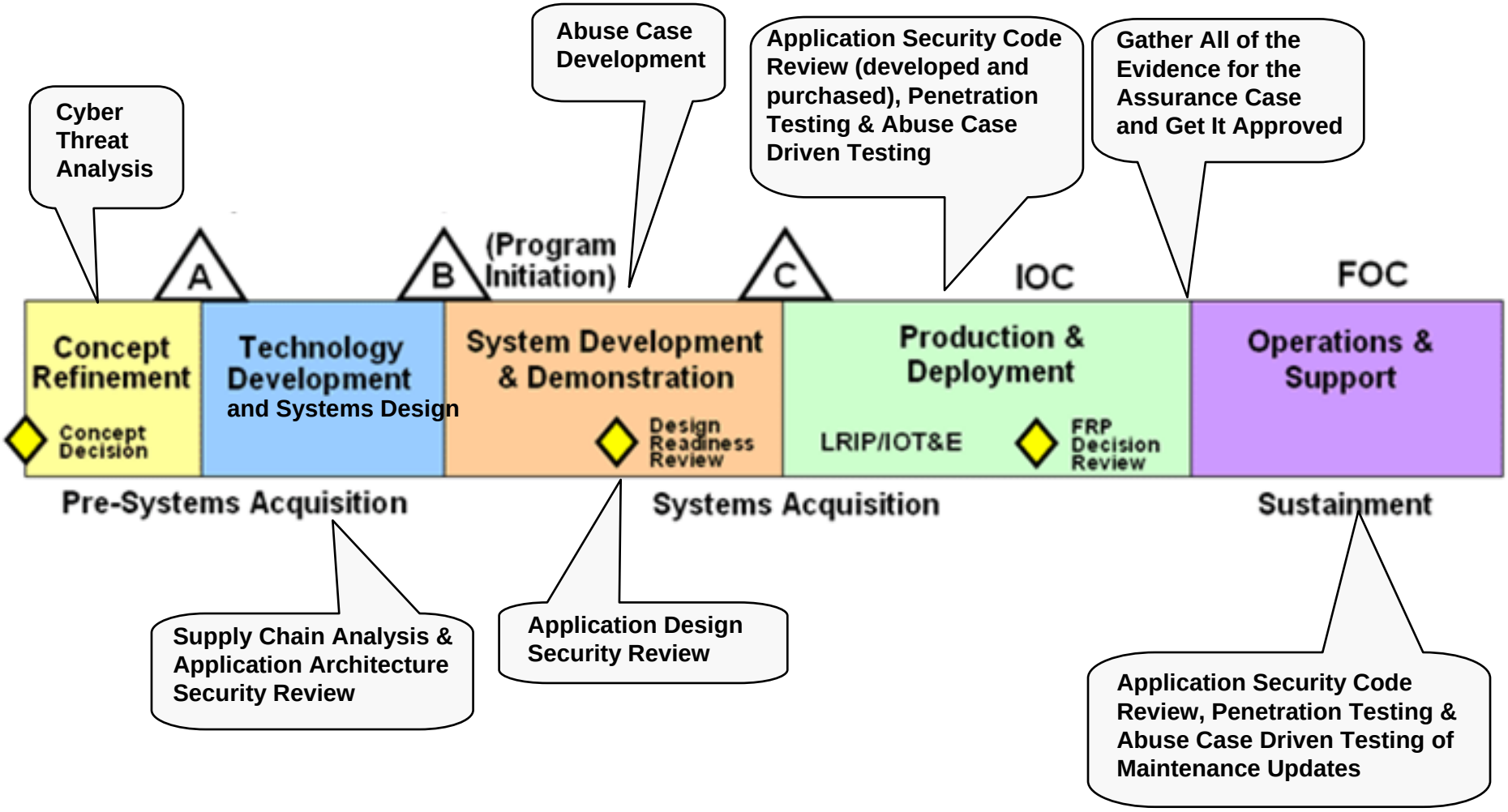
Using what we understand

- **Policy/practices for use & acquisition**
- **Composition of trust**
- **Hardware support**

***Multiple Sources:**

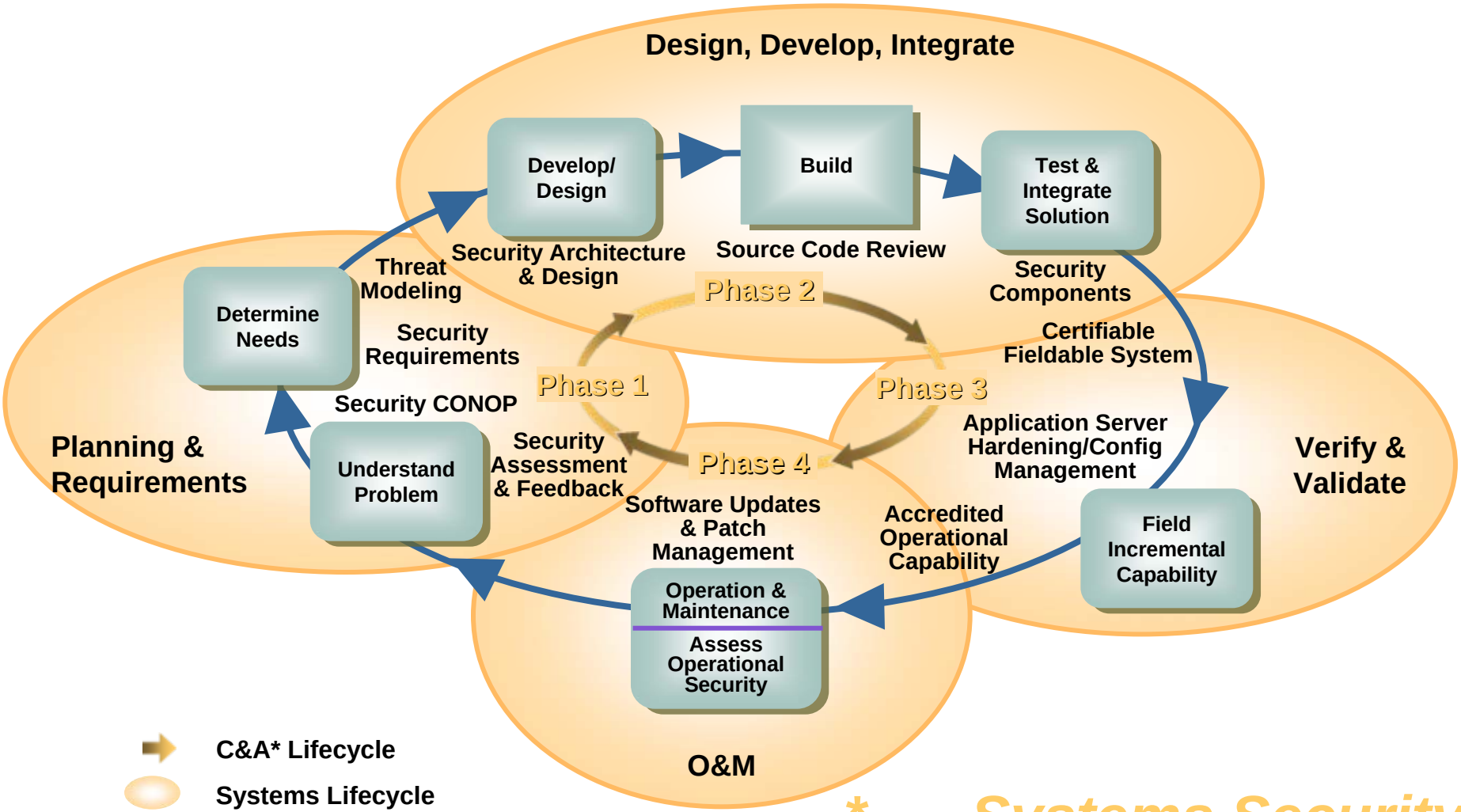
DHS/NCSD, OASD(NII)IA, NSA, NASA, JHU/APL

SwA and Systems Development (example)



*** Ideally Insert SwA before RFP release**

Integrating SwA into the Systems Engineering Lifecycle



* Systems Security Certification and

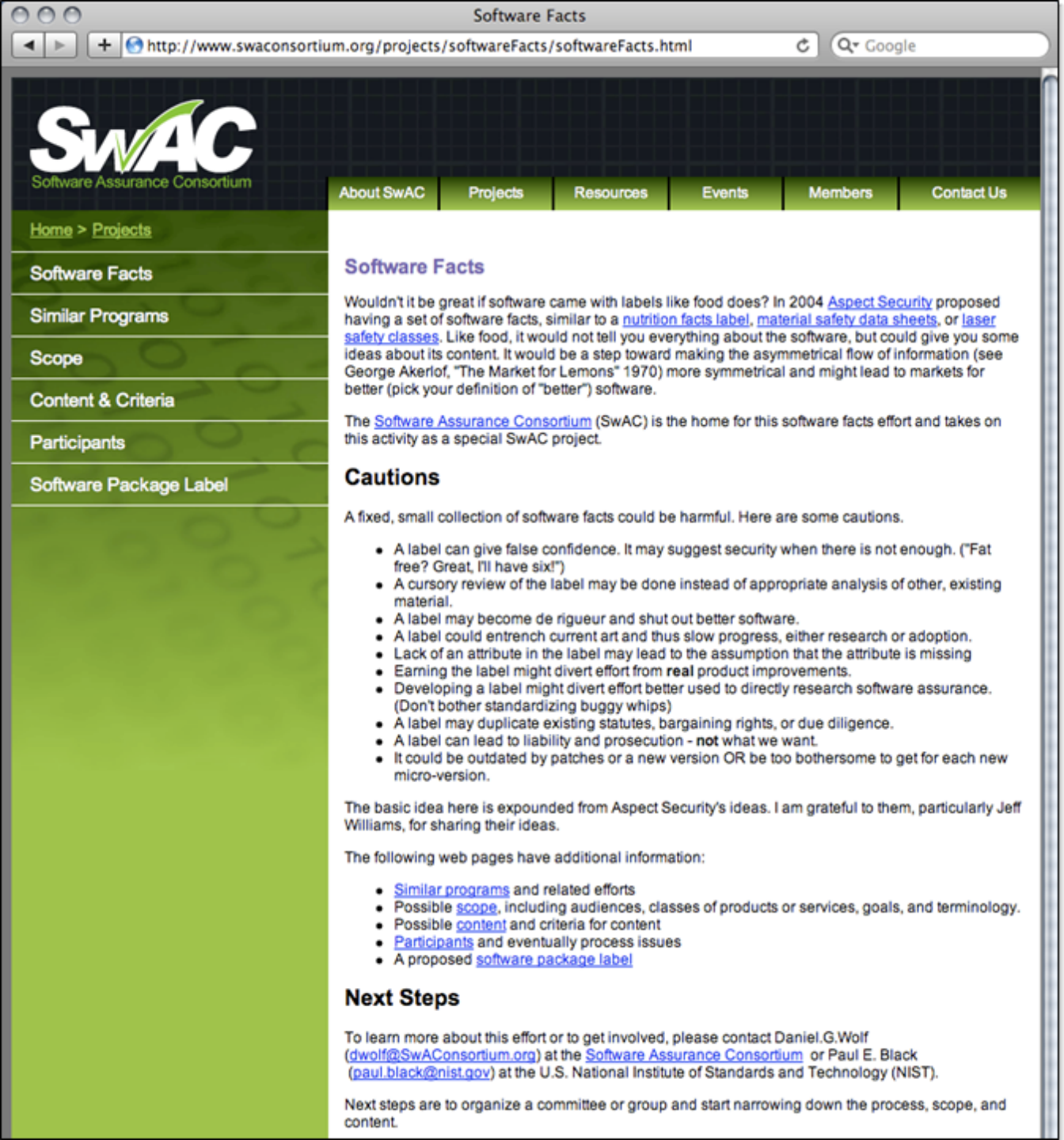
Software Assurance Lifecycle Considerations

- Define Lifecycle Threats/Hazards, Vulnerabilities & Risks
- Identify Risks attributable to software
- Determine Threats (and Hazards)
- Understand key aspects of Vulnerabilities
- Consider Implications in Lifecycle Phases:
  - Threats to: System, Production process, Using system
  - Vulnerabilities attributable to: Ineptness (undisciplined practices), Malicious intent, Incorrect or incomplete artifacts, Inflexibility
  - Risks in Current Efforts: Polices & Practices, Constraints

The Assurance Case/Argument – Requires Measurement

- Set of structured assurance claims, supported by evidence and reasoning, 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 and regulatory requirements**
  - **The configuration baseline**
  - **Identified hazards and threats and residual risk of each hazard and threat**
  - **Operational and support assumptions**

***Adopted from Paul Croll, ISO SC7 WG9 Editor for Systems and Software Assurance**



Software Facts

Name InvadingAlienOS
Version 1996.7.04
Expected number of users 15

Modules 5 483 Modules from libraries 4 102

No Severe Vulnerabilities Found
by methods that will be
out of date by Feb 2009

	% **Weaknesses Found**
Cross Site Scripting 22	76%
Reflected 12	41%
Stored 10	34%
SQL Injection 2	7%
Buffer overflow 5	17%
Total Security Mechanisms 284	100%
Authentication 15	5%
Access control 3	1%
Input validation 230	81%
Encryption 3	1%
AES 256 bits, Triple DES	

Report security flaws to: ciwnmcyi@mothership.milkyway

Total Code 3.1415×10^{9} function points	100%
C 1.1×10^{9} function points	35%
Ada 2.0415×10^{9} function points	65%

Level W static analysis run on 42% of code

Test Material 2.718×10^{6} bytes	100%
Data 2.69×10^{6} bytes	99%
Executables 27.18×10^{3} bytes	1%
Documentation 12 058 pages	100%
Tutorial 3 971 pages	33%
Reference 6 233 pages	52%
Design & Specification 1 854 pages	15%

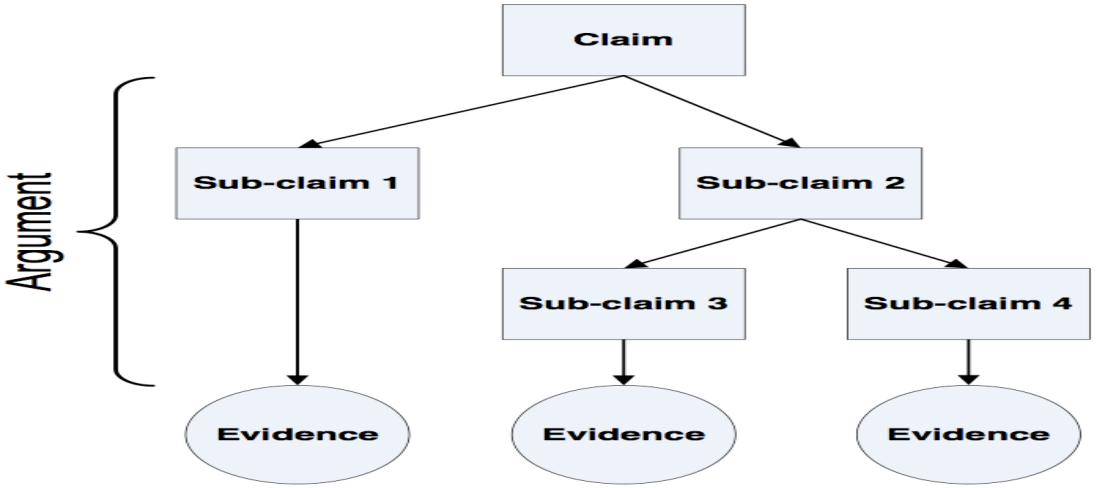
Libraries: Sun Java 1.5 runtime, Sun J2EE 1.2.2, Jakarta log4j 1.5, Jakarta Commons 2.1, Jakarta Struts 2.0, Harold XOM 1.1rc4, Hunter JDOMv1

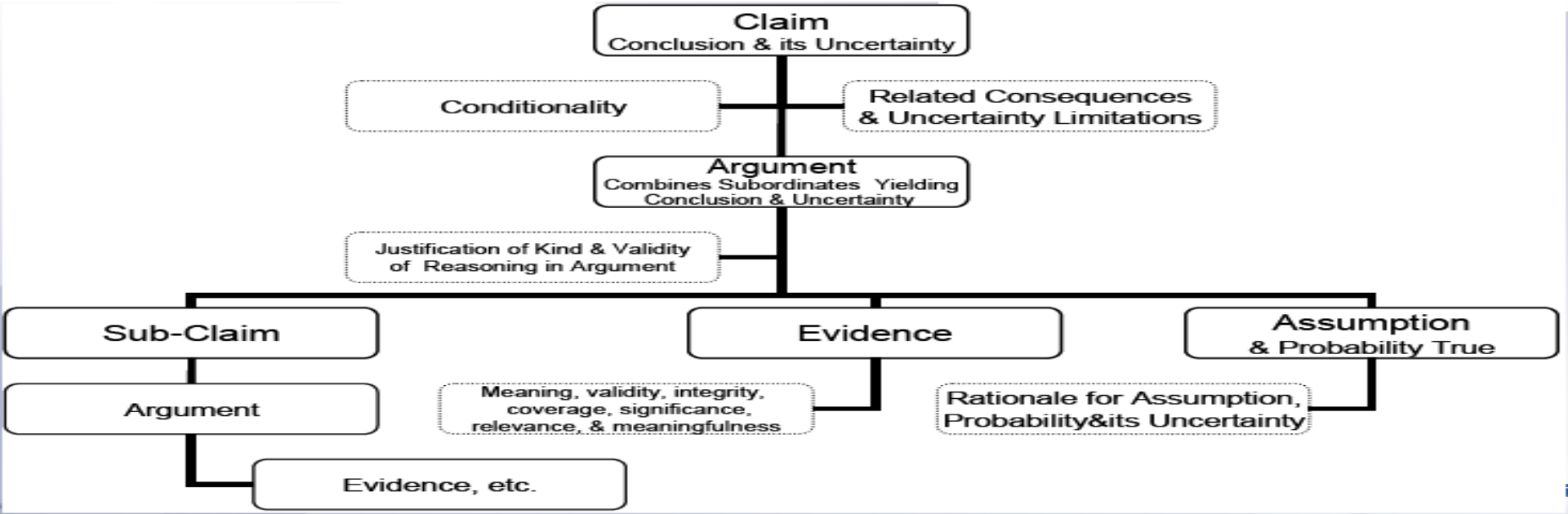
Compiled with gcc (GCC) 3.3.1

Stripped of all symbols and relocation information.

ISO/IEC 15026: Systems & Software Assurance

15026 Part 2: The Assurance Case (Claims-Evidence-Argument)

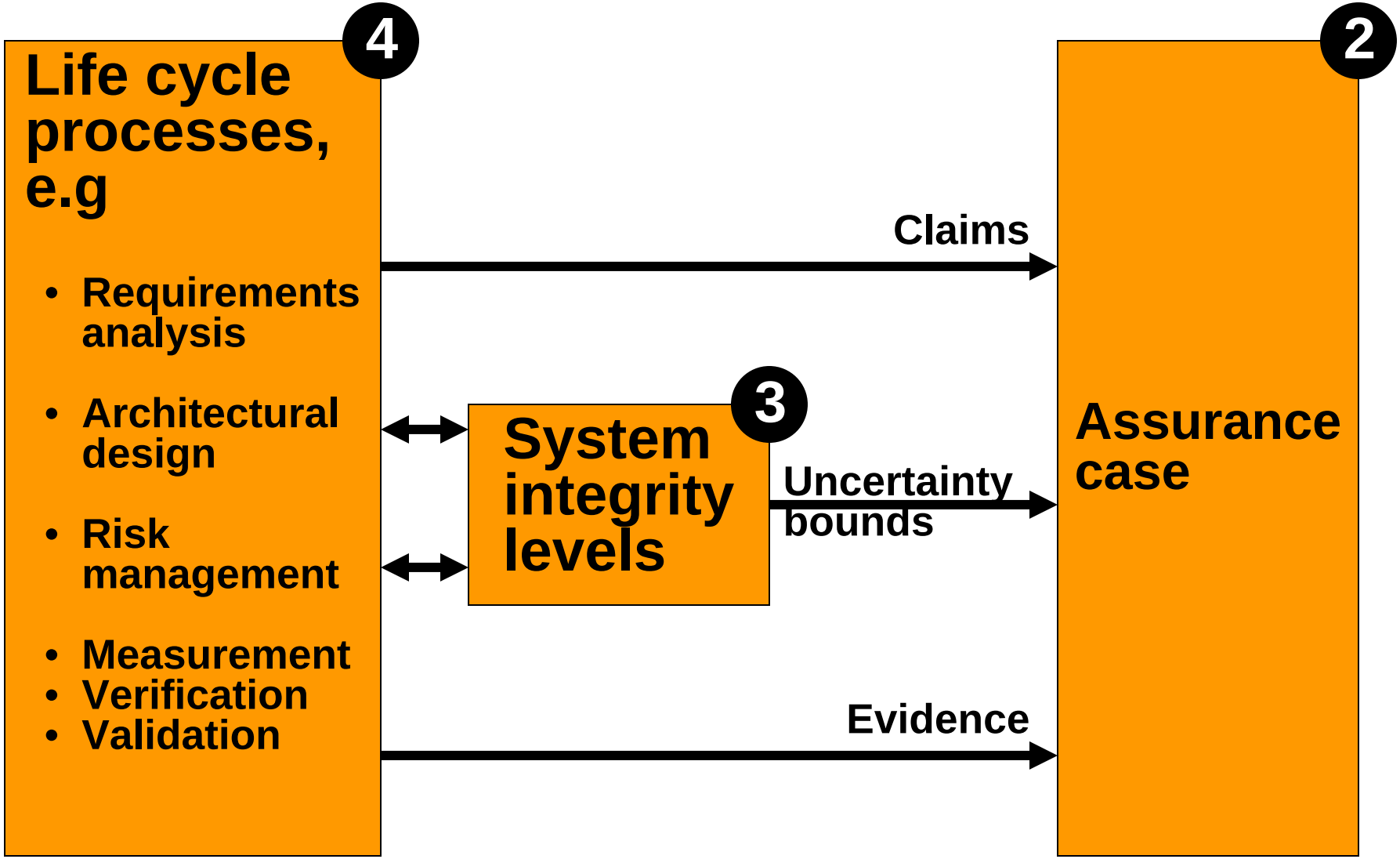




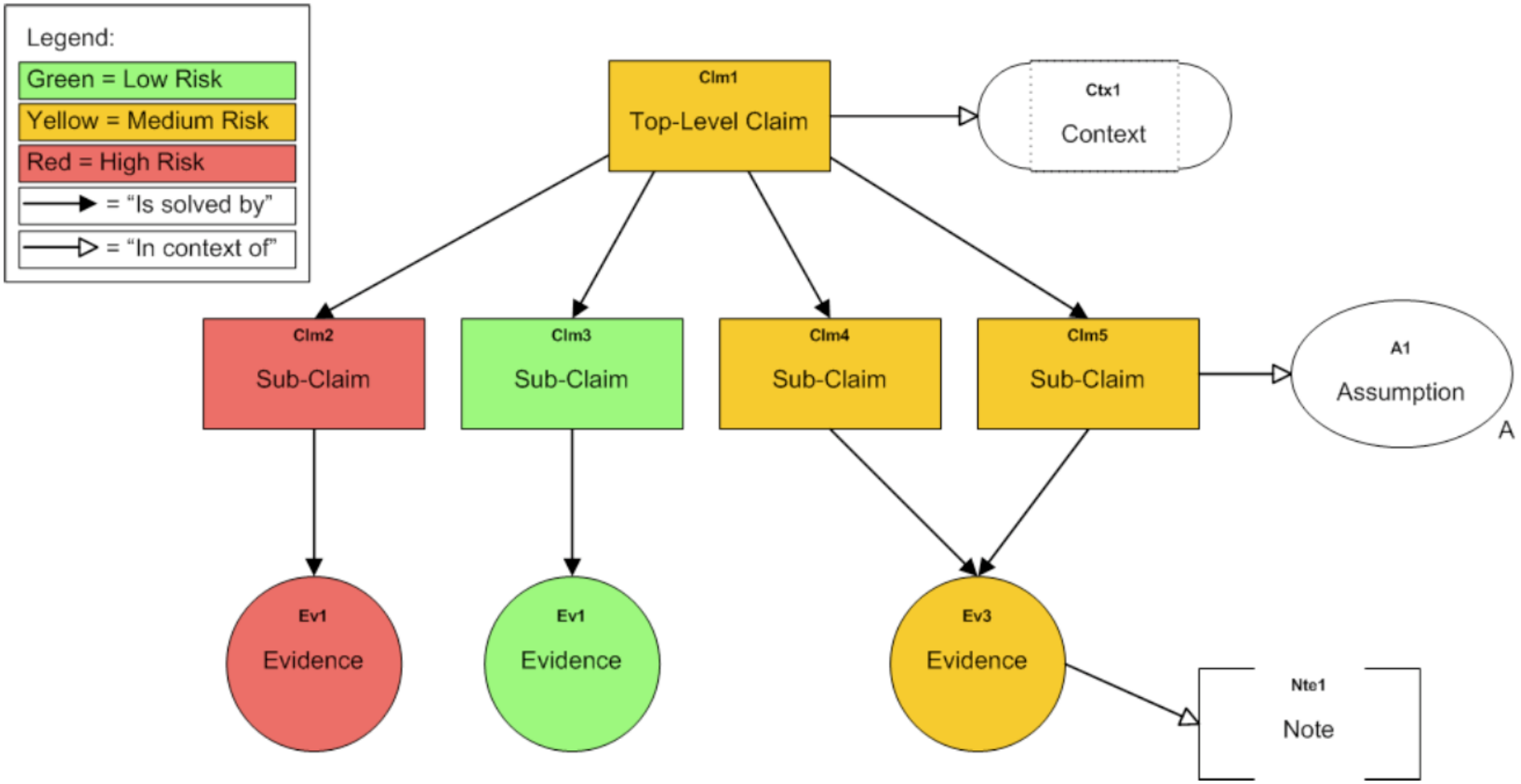
ISO/IEC 15026: A Four-Part Standard

- Planned parts:
  - **15026-1: Concepts and vocabulary (initially a TR2 and then revised to be an IS)**
  - **15026-2: Assurance case (including planning for the assurance case itself)**
  - **15026-3: System integrity levels (a revision of the 1998 standard)**
  - **15026-4: Assurance in the life cycle (including project planning for assurance considerations)**
- Possible additional parts as demand requires and resources permit, e.g.
  - **Assurance analyses and techniques**
  - **Guidance documents**

ISO/IEC 15026: Examples of relationships among parts

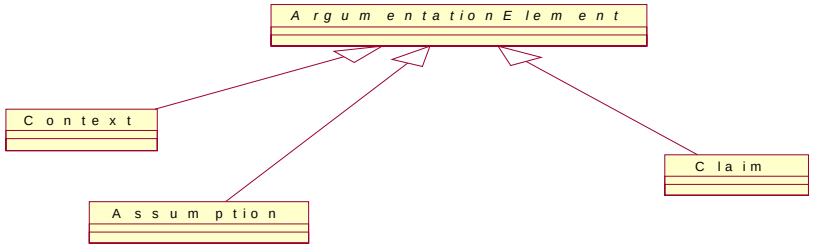


Safety Cases Based on Assurance Cases – Claims-Evidence-Argument in Use for <10 Years

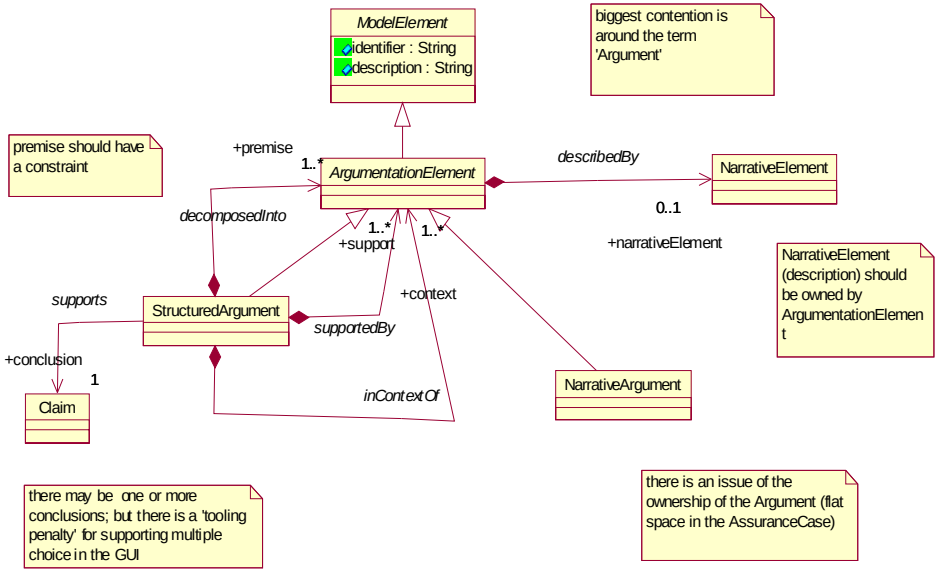


The Assurance Case/Argument: OMG Evidence and Claims/Arguments Standards

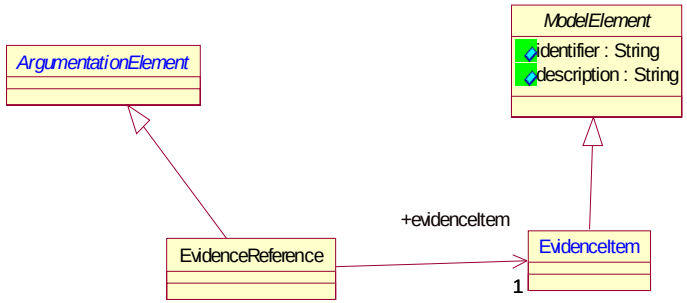
ARM:Claims

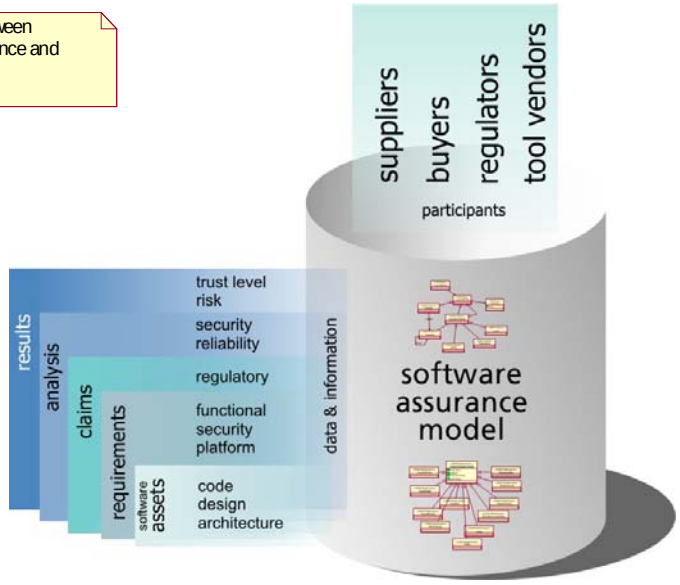


ARM:Arguments

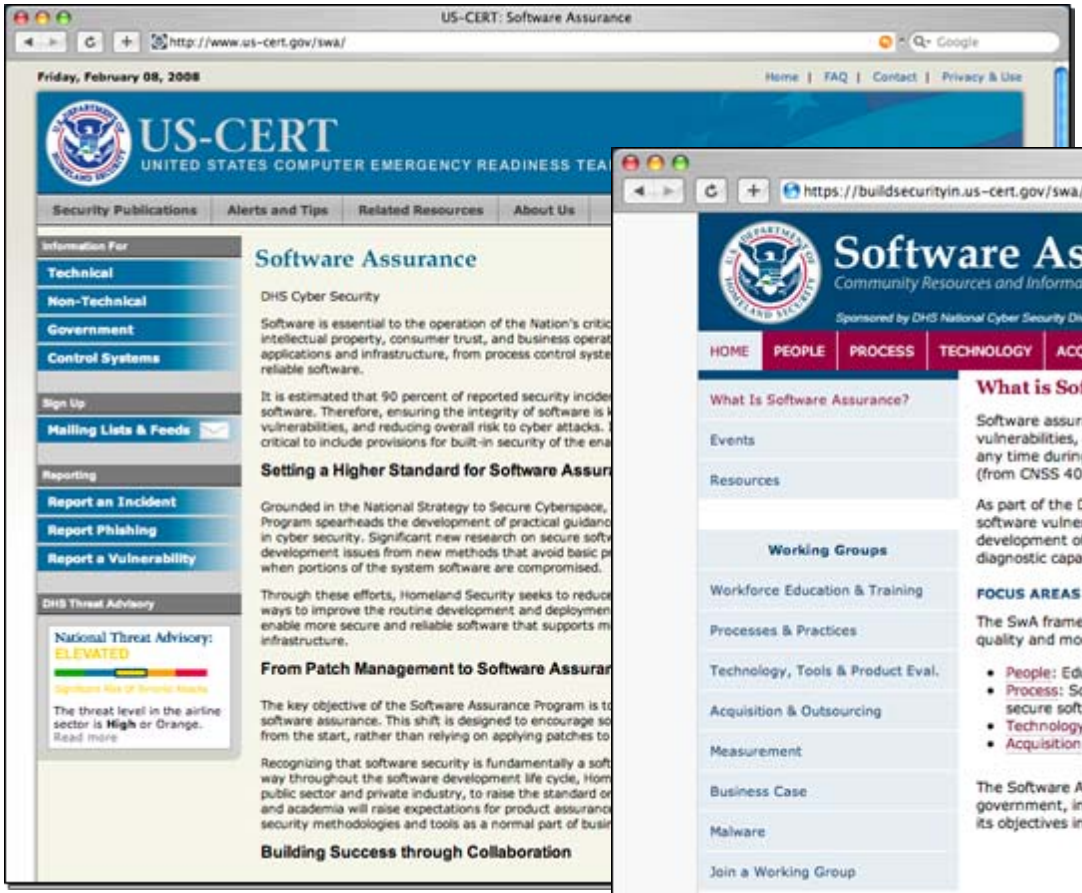


SAEM: Evidence





DHS's Build Security In and SwA Websites

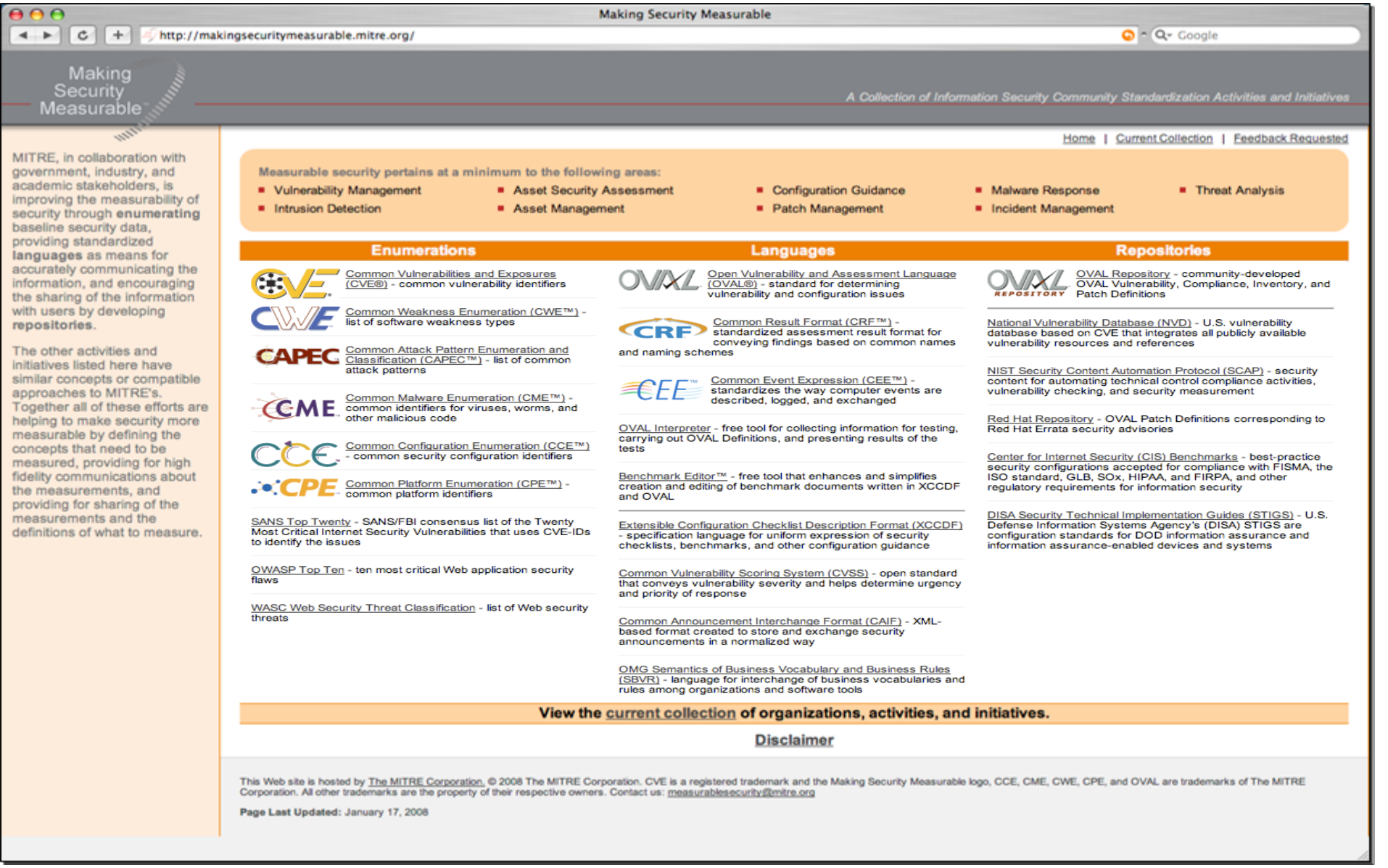


www.us-cert.gov/swa/

buildsecurityin.us-cert.gov/swa/

Homeland Security

[makingsecuritymeasurable.mitre.org]




Questions?