



Homeland
Security



Software Assurance (SwA) Automation...

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

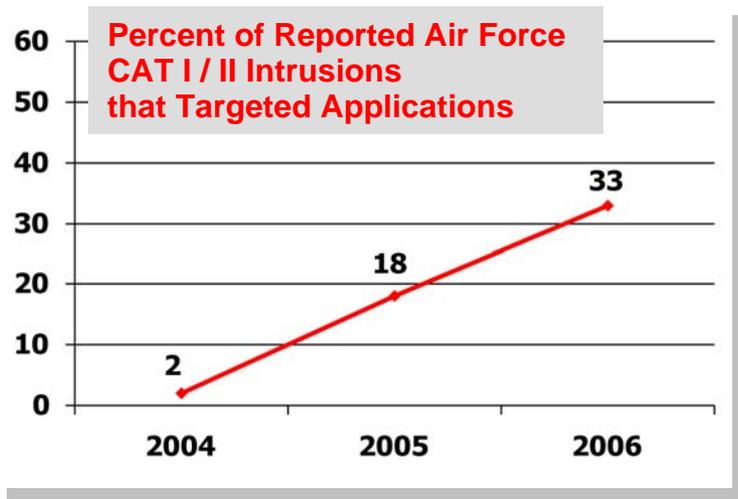
Making
Security
Measurable™



MITRE

110011010111001010110
0101010101010101010
0010100111010010110101
010110100011101010100
110011010111001010110
101011101000101001010
10101011101001010100
101011010110100101010

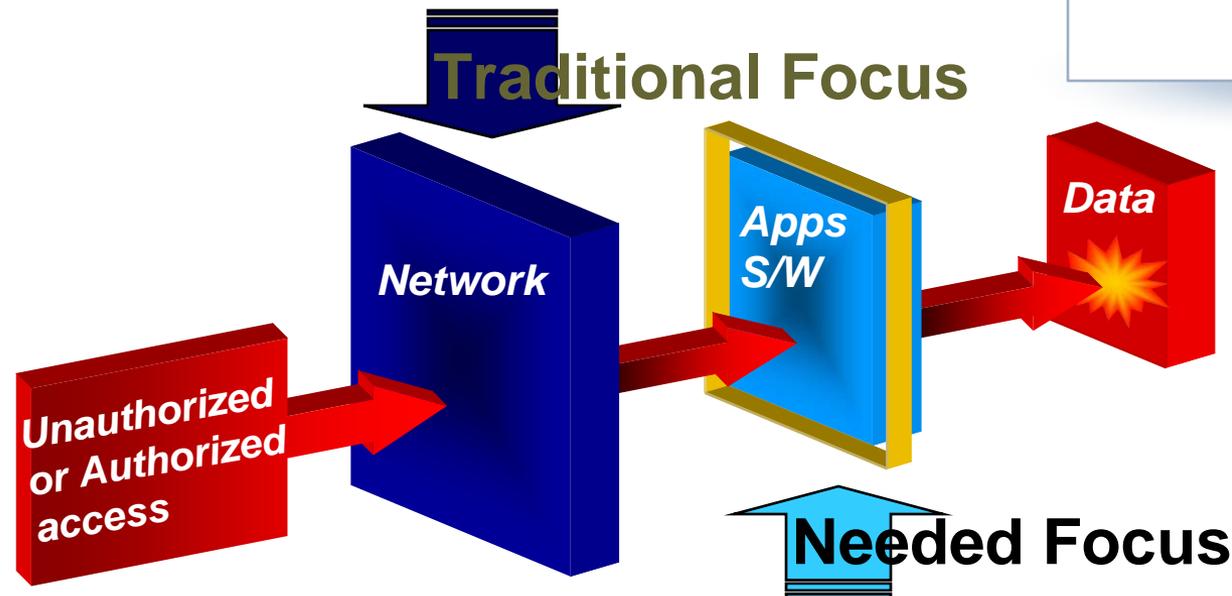
Application Security Wake-up Call



"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);

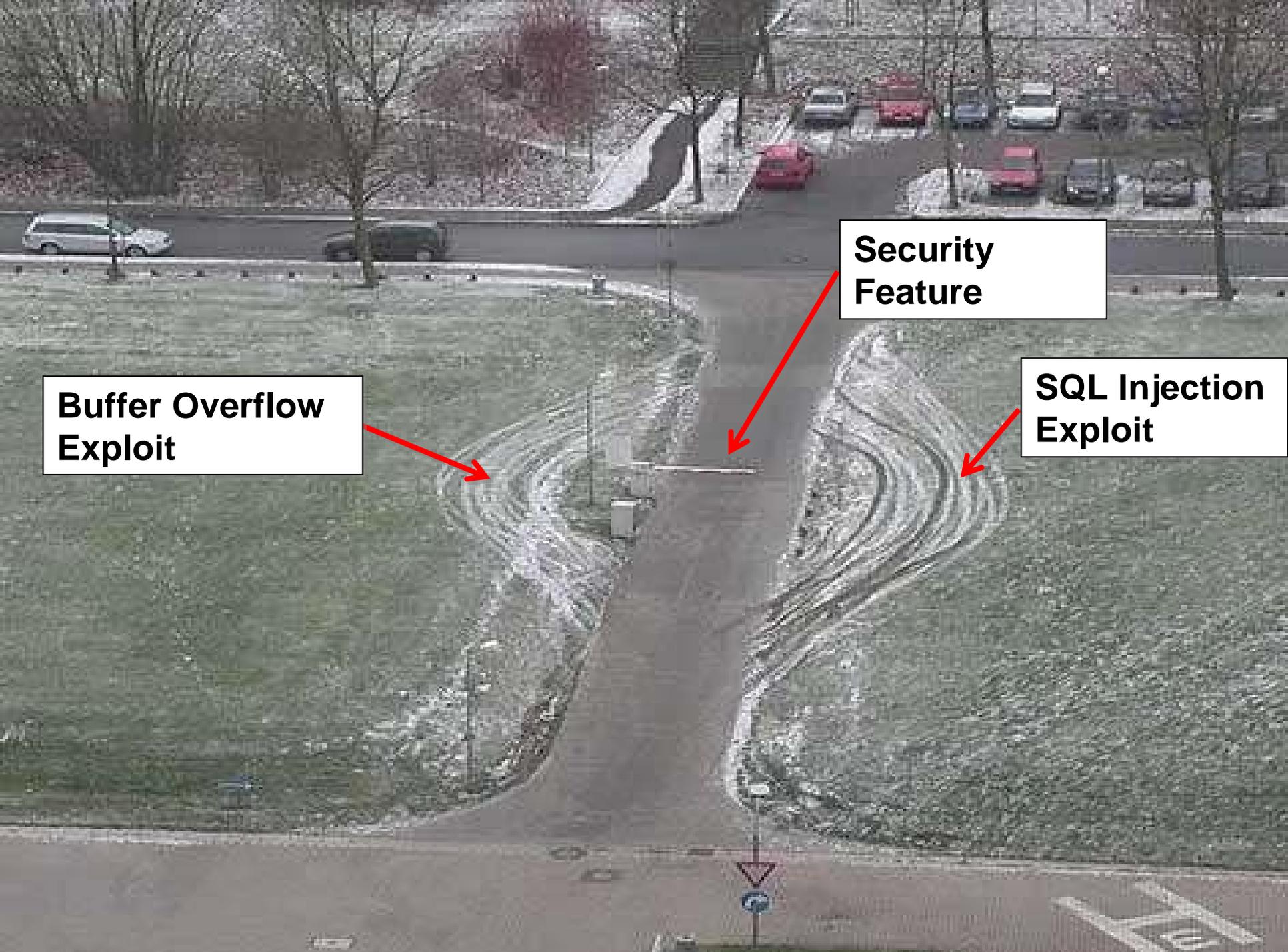
... exception when numeric_error =>
  calculate_vertical_veloc();
  when others => use_irs1();
  end;
end irs2;
```

- **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 -





**Buffer Overflow
Exploit**



**Security
Feature**



**SQL Injection
Exploit**

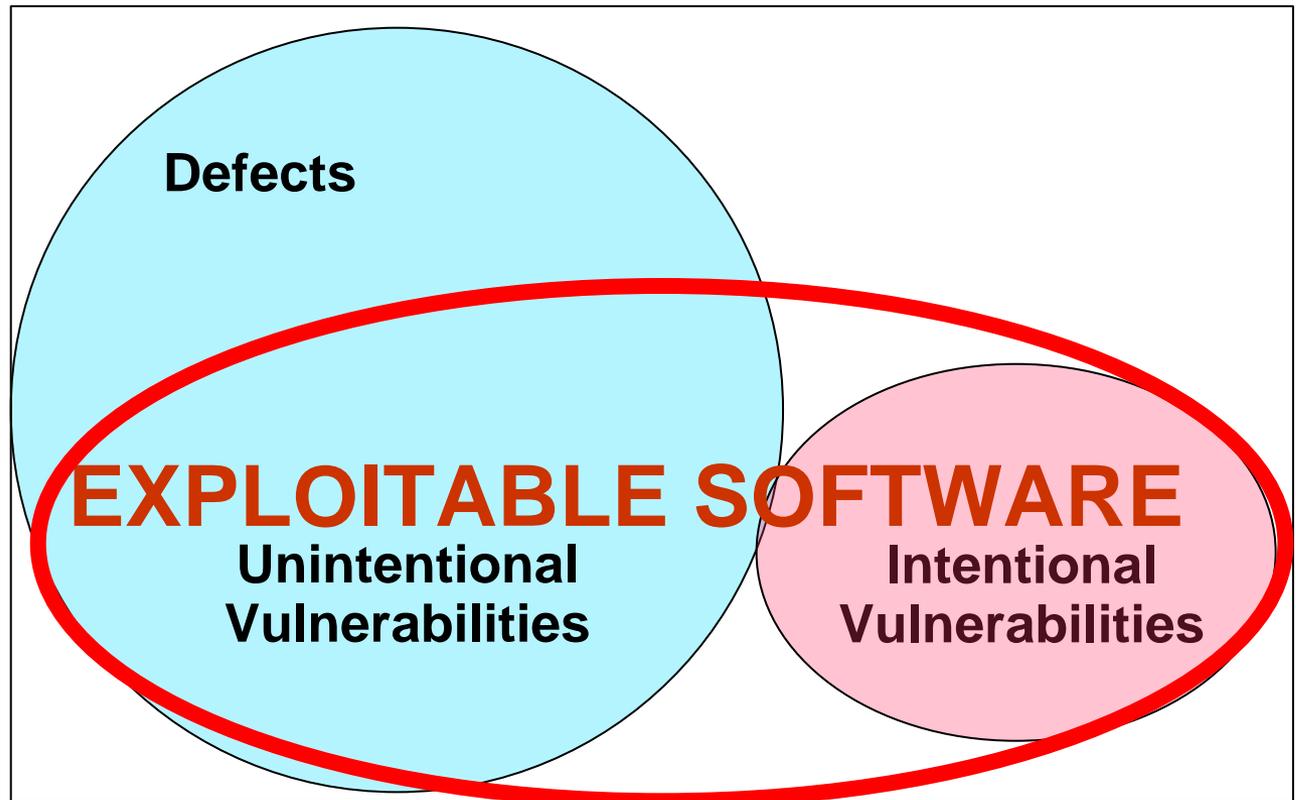


Exploitable Software Weaknesses (a.k.a. Vulnerabilities)

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.

**D
E
P
L
O
Y
E
D** **S
O
F
T
W
A
R
E**



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 discussion

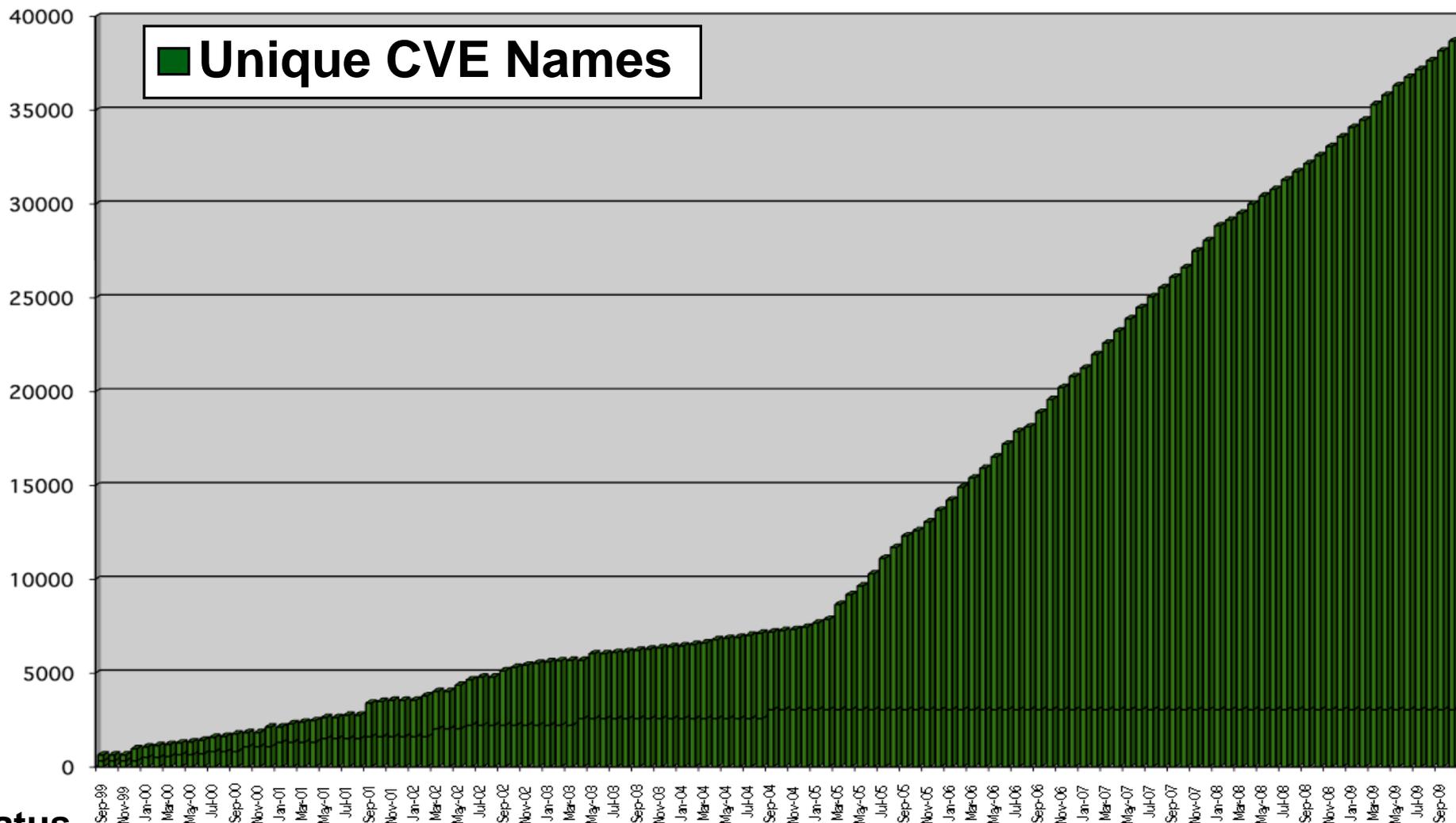


Software Vulnerabilities

- λ Serve as a primary point of entry that Attackers use to gain access to systems and/or data
- λ Expose business/mission systems to compromise
- λ 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**



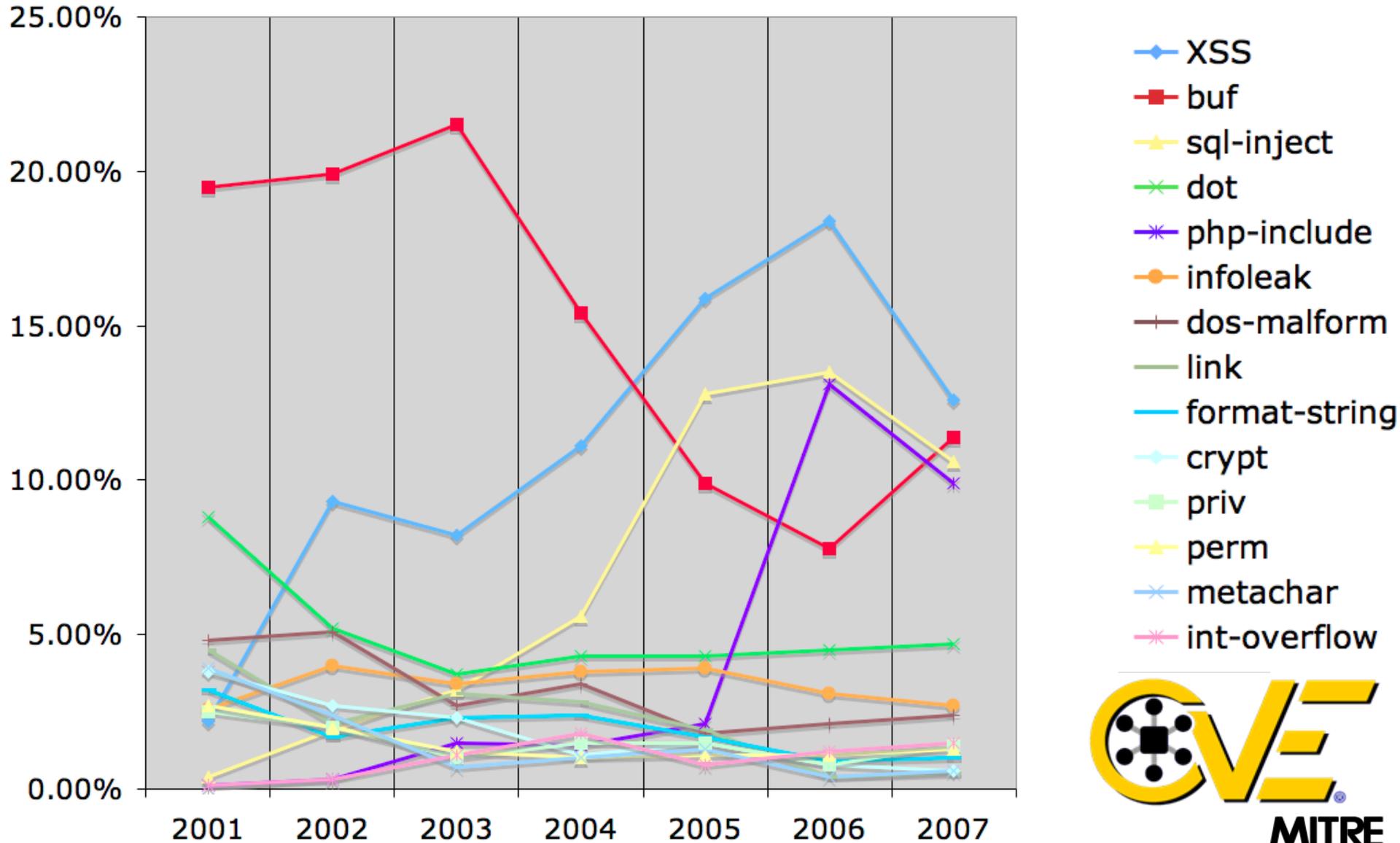
Publicly Known Vulnerabilities in “Packaged Software” (CVE) Growth



Status
(as of Oct 14, 2009)

• **38,839** unique CVE names

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



Removing and Preventing the Vulnerabilities Requires More Specific Definitions...CWEs

- ◆ XSS
- buf
- ◆ sql-inject
- ✕ dot
- ✱ php-include
- infoleak
- dos-malform
- link
- format-string
- ◆ crypt
- priv
- ◆ perm
- ◆ metachar
- ✱ int-overflow

- 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)

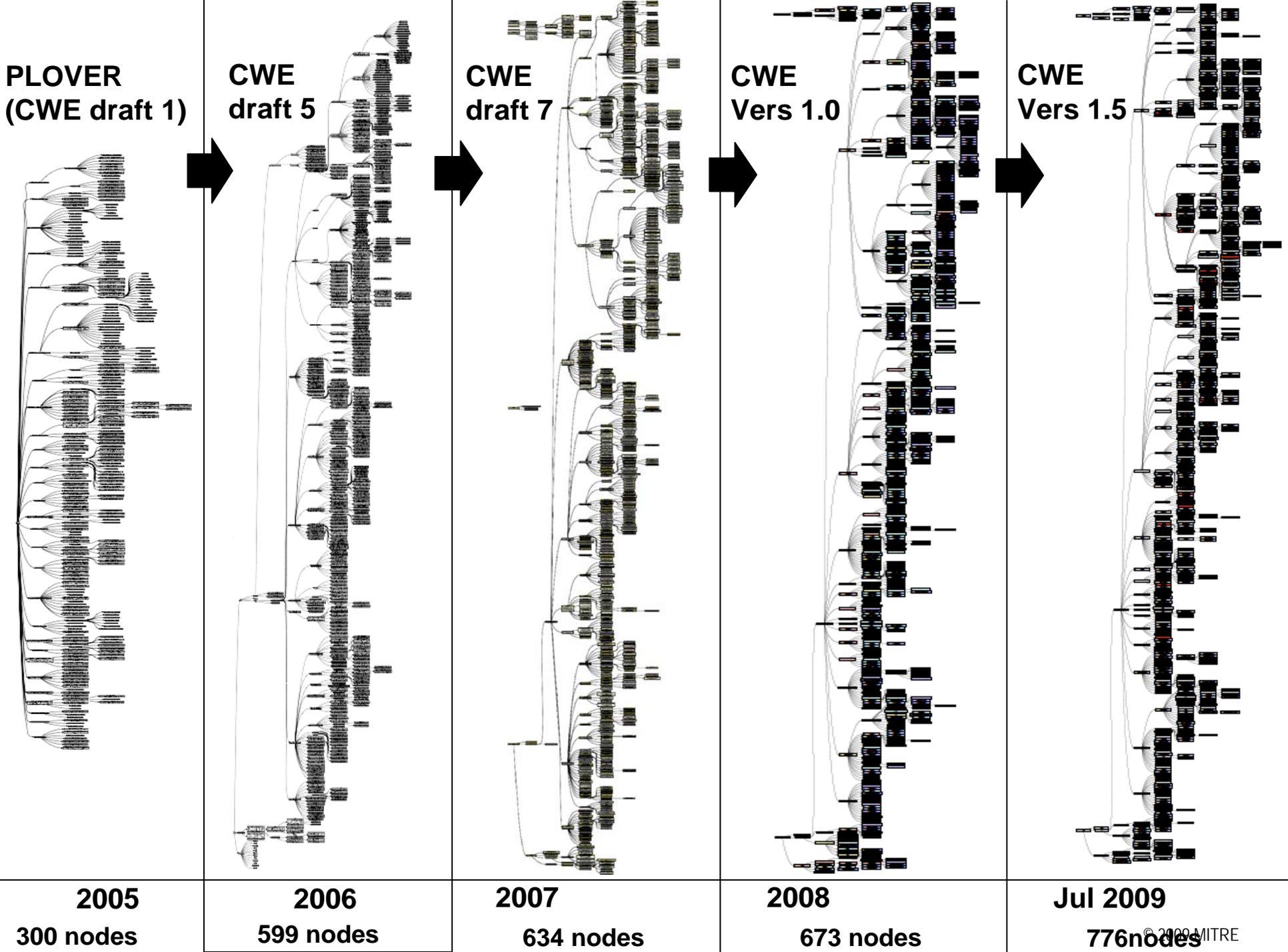
- Path Traversal (22)
- Relative Path Traversal (23)
 - Path Traversal: '\..\filename' (29)
 - Path Traversal: '\dir..\filename' (30)
 - Path Traversal: 'dir..\filename' (31)
 - Path Traversal: '...' (Triple Dot) (32)
 - Path Traversal: '....' (Multiple Dot) (33)
 - Path Traversal: '..../' (34)
 - Path 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

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



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



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

The screenshot shows a web browser window displaying the SANS Institute website. The page title is "The Top 25 Most Dangerous Programming Errors". The URL in the address bar is "http://www.sans.org/resources/top25/". The page features a navigation menu with links for "training", "certification", "resources", "vendor", "portal", "storm center", "college", "developer", and "about". The main content area is titled "The Top 25 Most Dangerous Programming Errors" and includes a sub-heading "Giving CIOs a Way to Measure the Security of the Software They Buy and Build". The text describes the collaboration between SANS Institute and MITRE to create a list of the 25 most significant programming errors that can lead to serious software vulnerabilities. It mentions that the list will be published in Washington DC on December 10, 2008, and will be provided to application software security testing vendors. A sidebar on the right contains a red banner for "Secure Coding in Java/JEE" with the text "Developing Defensible Applications" and "The only course covering the key elements of secure Java application development".

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



Attack Pattern 7
ID

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

Google Analytics

ramartin@mitre.org | Settings | My Account | Help | Sign Out

Analytics Settings | View Reports:

My Analytics Accounts:

Dashboard

› Saved Reports

Visitors

Traffic Sources

Content

Goals

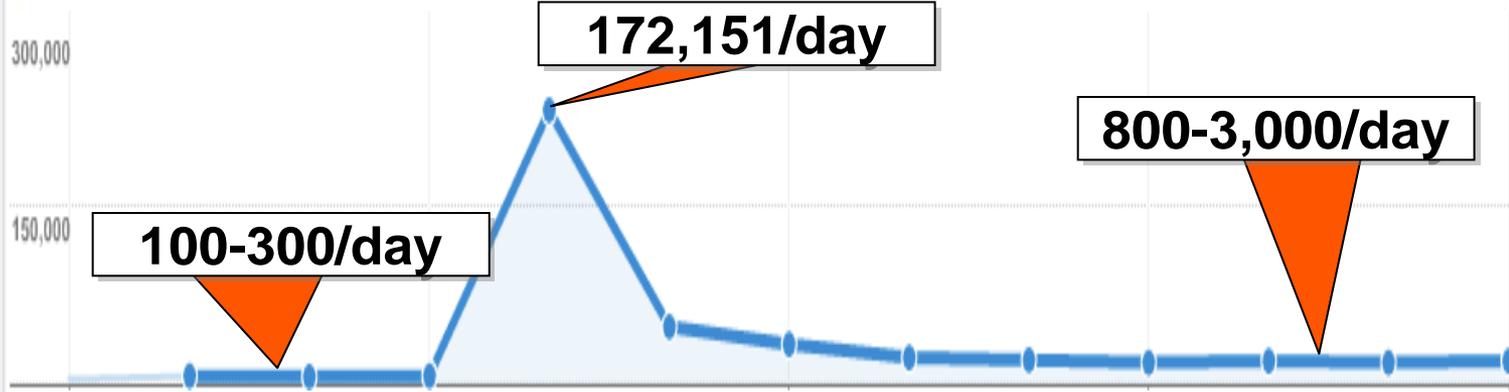
Custom Reporting Beta

Export

Beta Advanced Segments:

Dashboard

Oct 1, 2008 – Oct 1, 2009



Printable PDFs of Entire CWE Now Available



CWE Version 1.4

Edited by:
Steven M. Christey, Conor O. Harris, and Janis E. Kenderdine

Project Lead:
Robert A. Martin

A collage of overlapping PDF pages from the CWE 1.4 dictionary. The pages shown include:

- CWE Version 1.4 Table of Contents**: A list of page numbers from 1 to 53, with a Roman numeral 'iii' at the bottom.
- CWE-1: Location**: A page with a 'Status: Incomplete' header, a 'Good Code' section, and a table with columns 'W' and 'Page'. The table lists items 699, 700, and 701.
- CWE-1: Location**: A page with a 'Status: Draft' header, a 'Bad Code' section, and a table with columns 'W' and 'Page'. The table lists items 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000.
- CWE-69: Failure to Preserve SQL Query Structure (SQL Injection)**: A page with a 'Bad Code' section and a table with columns 'W' and 'Page'. The table lists items 699, 700, and 701.
- CWE-1: Location**: A page with a 'Status: Draft' header, a 'Bad Code' section, and a table with columns 'W' and 'Page'. The table lists items 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000.
- CWE-69: Failure to Preserve SQL Query Structure (SQL Injection)**: A page with a 'Bad Code' section and a table with columns 'W' and 'Page'. The table lists items 699, 700, and 701.
- CWE-1: Location**: A page with a 'Status: Draft' header, a 'Bad Code' section, and a table with columns 'W' and 'Page'. The table lists items 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000.



The Security Development Lifecycle

Welcome to MSDN Blogs [Sign in](#) | [Join](#) | [Help](#)

SEARCH

[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-2008-1844](#).

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 Common 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...

CWE-732: Insecure Permission Assignment for Critical Resource

I've already touched on this critical issue here, but review all windows and ACLs on all objects you create in the system configuration files such as Windows registry, in the case of Windows Vista and Linux, change any default ACLs in the system or registry unless you need to weaken the ACL.

CWE-330: Use of Insufficiently Random Values

Identify all the random non-predictable values used in your code and determine which, if any, require a pseudo-random algorithm. Make sure the code generating random numbers is cryptographically random and not a deterministic pseudo-random generator. The C routine `rand()` based using functions like `rand()` is, but not for cryptography.

CWE-250: Execution with Unnecessary Privileges

Identify all processes that run part of your solution and determine what privileges they need to operate correctly. If a process runs as root (or Linux, Unix, Mac OS X) or system (Windows), you should, "Why?" Because the account is usually called by the code, most perform a privileged operation, but someone you don't know why it runs away other than "That's the way it's always run." If the code needs to operate at high privilege, keep the time span within which the code is high privilege as small as possible—for example, using a port before 1024 in a Linux application requires the code to run as root, but after that,

Basic Training

particular that do the and path to the file and permission and ACLs on all objects you create in the system configuration files such as Windows registry, in the case of Windows Vista and Linux, change any default ACLs in the system or registry unless you need to weaken the ACL.

CWE-428: Untrusted OS

OS vendors created the file system directory structure, which problems if the had a weak path. A user's command, no guarantee that won't be a search or search from a process, environment is usually a in a path, but this is unnecessary.

CWE-319: Cleartext Transmission of Sensitive Information

Sensitive data must always be protected at rest and while in the wire. The best solution to this vulnerability is to use a well-tested technology such as SSL/TLS or IPsec. Don't invent your own communication method and cryptographic scheme. This weakness is related to CWE-327 ("Use of a Broken or Risky Cryptographic Algorithm"), so make sure you aren't using weak 40-bit RC4 or shared-key IPsec.

CWE-94: Failure to Generate

It's common to see code ignore vulnerabilities in developer code that build a string dynamically and gives it to `eval()` or `exec()`. If the attacker controls the source string in any way, he or she can create a malicious payload. The simplest way to eradicate this kind of bug is to eradicate the use of `eval()`, but that could mean redesigning the application.

CWE-79: Cross-Site Scripting (XSS)

CWE-79 is the real bug that makes CWE-116 worse. In the past, we took XSS bugs lightly, but now we see worms that can exploit XSS vulnerabilities in social networks such as MySpace (for example, the Sassy worm). Also, search firms Web-related vulnerabilities has progressed substantially over the past few years, with new ways to attack systems regularly increased. For past XSS issues as defined by CWE-79, the best defense is to validate all incoming data. This has always been the right approach and will probably continue to be so for the foreseeable future. Developers can also add a layer of defense by encoding output derived from untrusted input (see CWE-116).

CWE-78: Failure to Preserve OS Command Structure

Many applications, particularly server applications, receive untrusted requests and use the data in them to interact with the underlying operating system. Unfortunately, this can lead to severe server compromise if the incoming data isn't analyzed—again, the best defense is to check the data. Also, running the potentially vulnerable application with low privilege can help contain the damage.

CWE-319: Cleartext Transmission of Sensitive Information

Sensitive data must always be protected at rest and while in the wire. The best solution to this vulnerability is to use a well-tested technology such as SSL/TLS or IPsec. Don't invent your own communication method and cryptographic scheme. This weakness is related to CWE-327 ("Use of a Broken or Risky Cryptographic Algorithm"), so make sure you aren't using weak 40-bit RC4 or shared-key IPsec.

CWE-682: Incorrect Calculation

Many buffer overruns in C and C++ code today are actually related to incorrect buffer- or array-size calculations. If an attacker controls one or more of the elements in a size calculation, he or she can

CWE-352: Cross-Site Request Forgery

Cross-site request forgery (also known as CSRF) vulnerabilities are a relatively new form of Web weakness caused, in part, by a bad Web application design. In short, this design doesn't verify that a request came from valid user code and is instead acting maliciously on the user's behalf. Generally, the best defense is to use a unique and unpredictable key for each user. Traditionally, verifying input doesn't mitigate this bug type because the input is valid.

CWE-362: Race Condition

Race conditions are timing problems that lead to unexpected behavior—for example, an application uses a filename to verify that a file exists and then uses the same filename to open that file. The problem is in the small time delay between the check and the file open, which attackers can use to change the file or delete or create it. The safest way to mitigate file system race conditions is to open the object and then use the resulting handle for further operations. Also, consider reducing the scope of shared objects—for example, temporary files should be local to the user and not shared with multiple user sessions. Correct use of synchronization primitives (mutexes, semaphores, critical sections) is similarly important.

CWE-642: External Control of Critical State Data

Unprotected state information such as private data or configuration, is subject to an attacker's attempt to protect it by using the appropriate control list (ACL) or permissions for persistent data and size of cryptographic devices, a hashed message authentication code (HMAC), or cookie data. You can use an HMAC per-session data as well.

CWE-73: External Control of Filename or Path

Attackers might be able to arbitrarily file data if they the data that's used in part or path name. It's critical

CWE-119: Failure to Constrain Memory Operations

The dreaded buffer overflow is a common type of vulnerability type, but more headaches than both. The best way to overcome this problem is to move away from C and C++ where it makes sense and use higher-level languages such as Ruby, C#, and so on. In case they don't offer solutions to memory, for C and C++, developers should "know their" functions in C runtime (for example, `strncpy`, `strncpy_s`, `strncpy_s_s`, `strncpy_s_l`, and `strncpy_s_l_s`) and use secure versions. Visual C++ may weak APIs to copy and you should strive to complete. Also, for static analysis can help find potential buffer overruns, operating-system-level, such as address space layout randomization and no-execute can help reduce the chance of buffer overruns is exploited.

CWE-642: External Control of Critical State Data

Unprotected state information such as private data or configuration, is subject to an attacker's attempt to protect it by using the appropriate control list (ACL) or permissions for persistent data and size of cryptographic devices, a hashed message authentication code (HMAC), or cookie data. You can use an HMAC per-session data as well.

CWE-73: External Control of Filename or Path

Attackers might be able to arbitrarily file data if they the data that's used in part or path name. It's critical

Basic Training

Editors: Richard Fent, r.fent@mitre.org; Michael Howard, mhoward@wisc.com

Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

In January 2009, MITRE and SANS issued the "2009 CWE/SANS Top 25 Most Dangerous Programming Errors" to help make developers more aware of the bugs that can cause security compromises

encoding Web-based output is a defense to ease the Developer doesn't detect and prevent malicious Web input for CWE-79 and CWE-23. However, the industry has seen many security bugs that could have been prevented if the developer had

<http://cwe.mitre.org/top25/>. I was one of the many people

READING OVER YOUR SHOULDER • DEALING WITH THE SUREMIGHT GIANT



Basic Training

68 Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

MICHAEL HOWARD

10001
01111
10001
11110
10001

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/aa512937.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/aliq/archive/2008/01/24/security-code-review-visual-studio.aspx>
- Security Code Review Guidelines, Adam Shostack; <http://www.verber.com/mark/cs/security/code-review.html>
- OWASP Top Ten; http://www.owasp.org/index.php/OWASP_Top_Ten_Project

CWE CAPEC

10001
01111
10001
11110
10001

Testing

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

Fuzz testing

Fuzz testing is a reliability and security testing technique that relies on submitting 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/>**



10001
01111
10001
11110
10001
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 Blitz, SAP AG | Steve Lipner, Microsoft Corp. |
| Jerry Cochran, Microsoft Corp. | Brad Minnis, Juniper Networks, Inc. |
| Matt Coles, EMC Corporation | Hardik Parekh, EMC Corporation |
| Danny Dhillon, EMC Corporation | Dan Reddy, EMC Corporation |
| Chris Fagan, Microsoft Corp. | Alexandr Selczynov, Nokia |
| Cassio Goldschmidt, Symantec Corp. | Janne Usilehto, Nokia |
| Wesley Higaki, Symantec Corp. | Antti Vihä-Sipliä, Nokia |

10001
01111
10001
11110
10001
SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

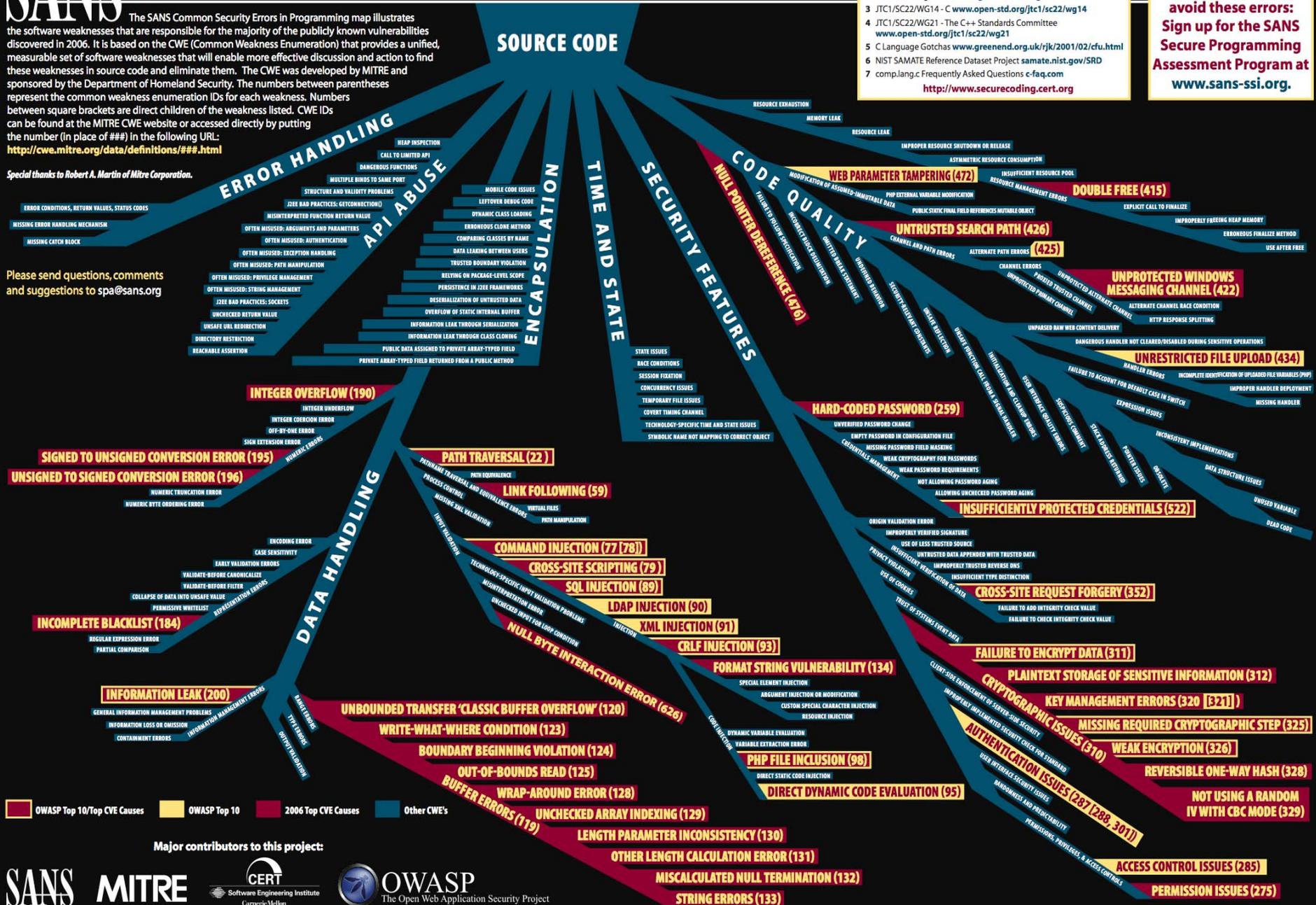
The SANS Common Security Errors in Programming map illustrates the software weaknesses that are responsible for the majority of the publicly known vulnerabilities discovered in 2006. It is based on the CWE (Common Weakness Enumeration) that provides a unified, measurable set of software weaknesses that 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. Numbers between square brackets 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.faq 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.



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

Special thanks to the CWE Team at MITRE.

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

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 - (695)

User Interface Errors

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

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 - (683)
- Incorrect Calculation of Buffer Size
- Integer Overflow or Wraparound
- Integer Underflow (Wrap or Wraparound)
- Off-by-one Error
- Divide By Zero

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 Structures (Cross-site Scripting) - (79)
- Improper Sanitization of Special Elements used in an SQL Command (SQL Injection) - (84)
- 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 Different Plane
- Argument Injection or Modification
- Improper Control of Resource Identifiers (Resource Injection)
- Improper Sanitization of Special Elements

Representation Errors

- Cleaning, 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 - (206)
- Cross-boundary Cleaning Information Leak
- Intended Information Leak
- Process Environment Information Leak
- Information Leak Through Debug Information
- Sensitive Information Uncleaned 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

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 ("Falling Open")
- Missing Custom Error Page

Pointer Issues

- Return of Pointer Value Outside of Expected Range
- Use of size() 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
- JZEE 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
- JZEE 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

Credentials Management

- Hard-Coded Password - (250)
- Unwhittled Password Change
- Missing Password Field Masking
- Weak Cryptography for Passwords
- Weak Password Requirements
- Not Using Password Aging
- Password Aging with Long Expiration
- Insufficiency Protected Credentials
- Weak Password Recovery Mechanism for Forgotten Password

Insufficient Verification of Data Authenticity

- Origin Validation Error
- Improper Verification of Cryptographic Signature
- Use of Least 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 - (230)
- Logging of Excessive Data
- Certificate Issues

Insufficient Encapsulation

Mobile Code Issues/Missing Custom Error Page

- Public (nonstatic) Method Without Final (Object Hijack)
- Use of Inner Class Containing Sensitive Data
- Critical Public Variable Without Final Modifier
- Download of Code Without Integrity Check - (484)
- Array Declared Public, Final, and Static (Analyze) 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

Security Features

Cryptographic Issues

- Key Management Errors
- Unwhittled 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 Flaky Cryptographic Algorithm - (327)
- Use of RSA Algorithm without OAEP

Permissions, Privileges, and Access Controls

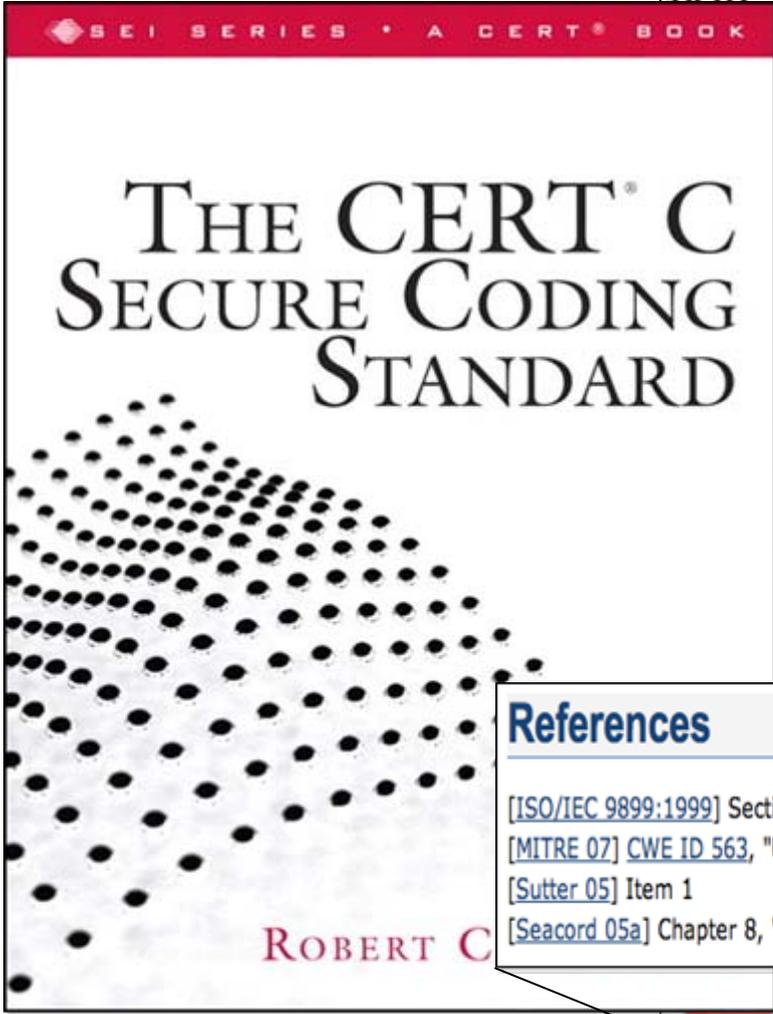
- Access Control (Authorization) Issue - (184)
- Permission Issues
- Incorrect Default Permissions
- Insecure Inherited Permissions
- Insecure Privileged Assigned Permissions
- Incorrect Execution-Assigned Permissions
- Improper Handling of Insufficient Permissions or Privileges
- Improper Preservation of Permissions
- Expose of 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

- Reliance on Package-level Scope
- JZEE Framework Saving Unserializable Objects to Disk
- Serialization 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



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

Home > Practices > C++ > Miscellaneous (MSC) > MSC00-CPP. Compile cleanly at high warning levels

C++ Secure Coding Practices

MSC00-CPP. Compile cleanly at high warning levels

Added by [Justin Roper](#), last edited by [Justin Roper](#) on Oct 08, 2008 ([view changes](#)) ([SHOW COMMENT](#))

Labels: [uninformative](#) [example-code](#)

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-EX3: 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 [substandard](#) 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

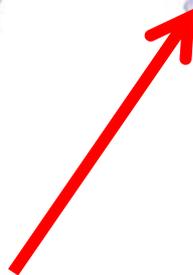
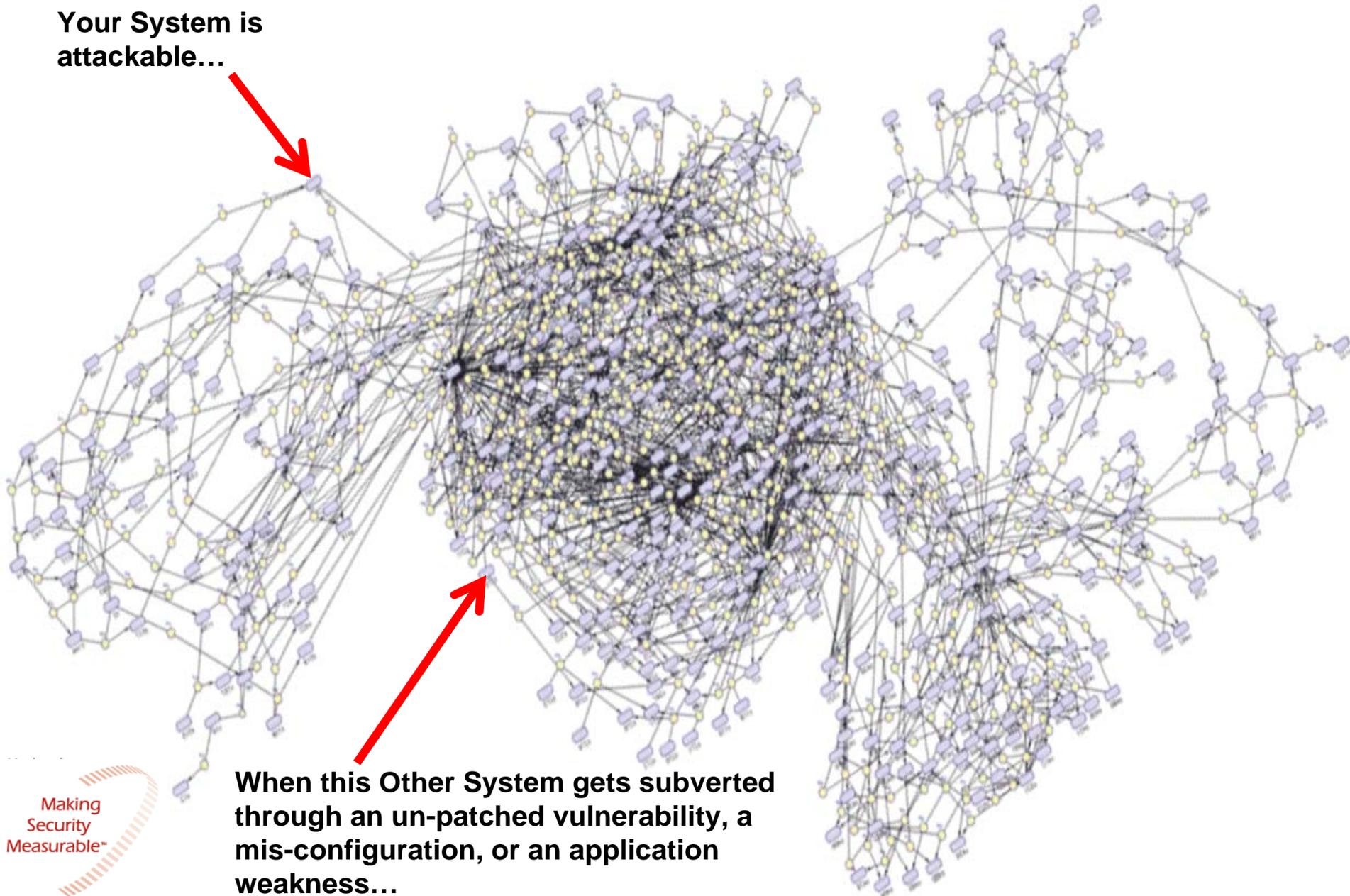
US-CERT

Go to <http://cwe.mitre.org/data/definitions/570.html>

© 2009 MITRE

Today Everything's Connected

Your System is
attackable...

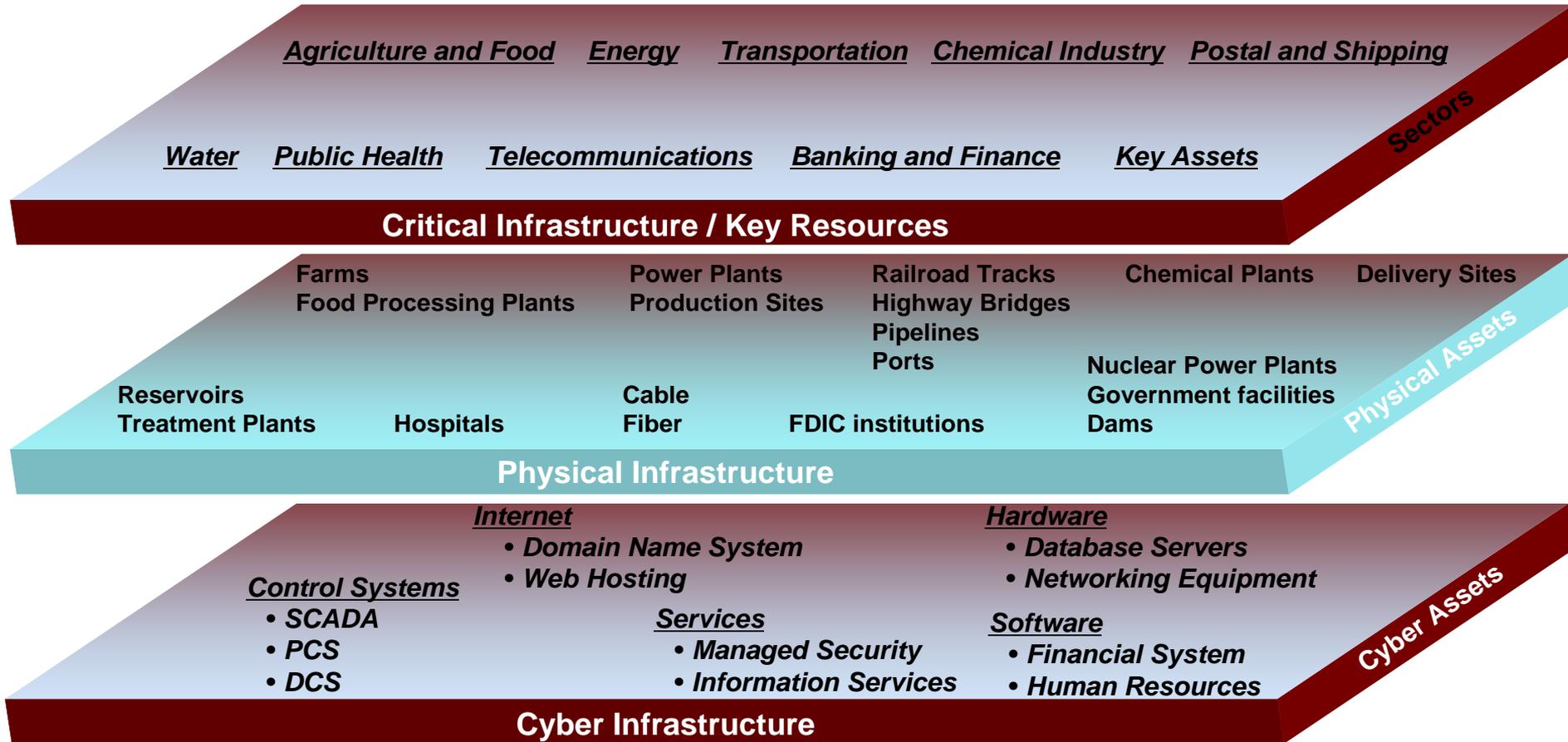


When this Other System gets subverted
through an un-patched vulnerability, a
mis-configuration, or an application
weakness...



Making
Security
Measurable™

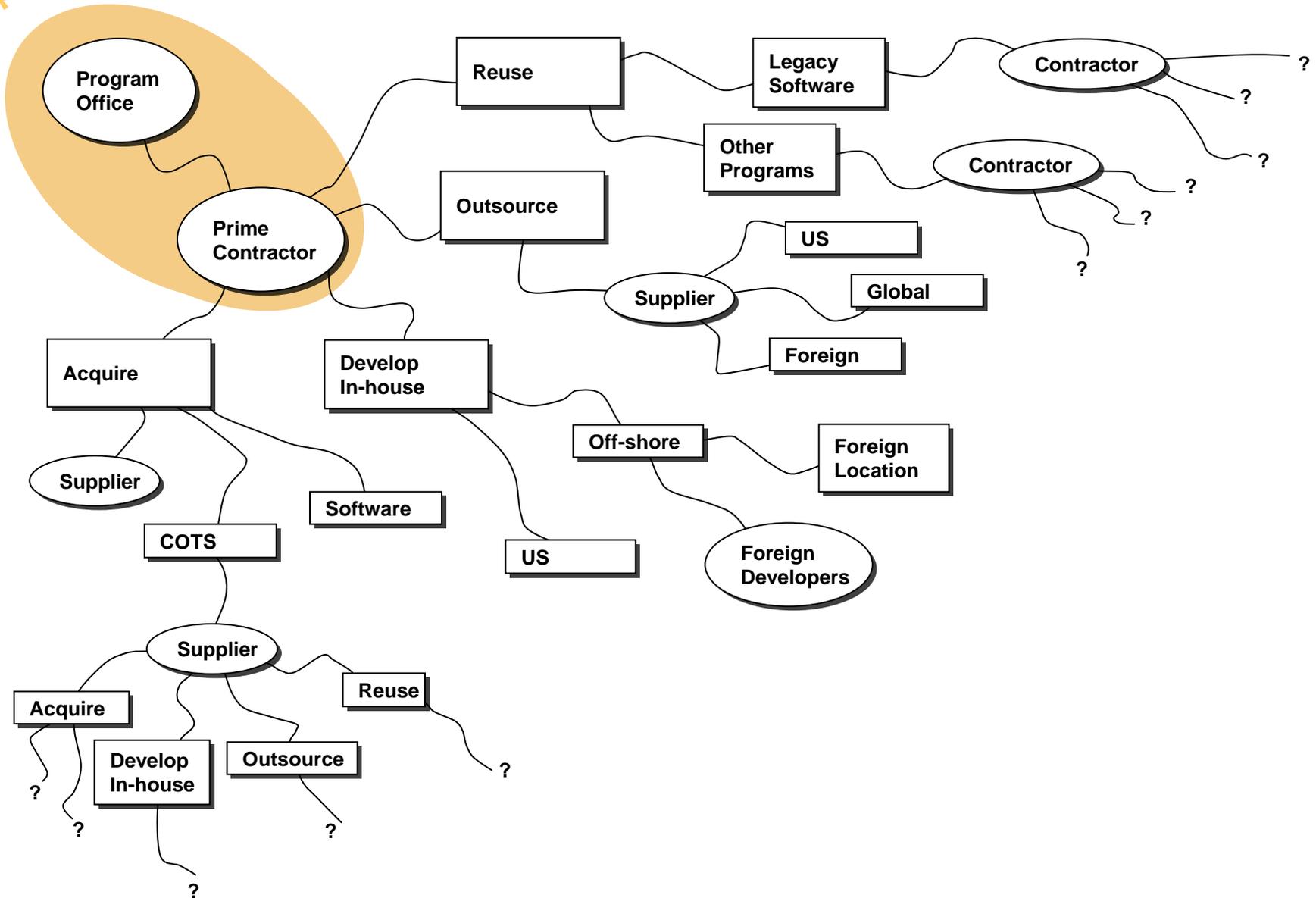
Cyberspace & physical space are increasingly intertwined and software controlled/enabled



Need for secure software applications

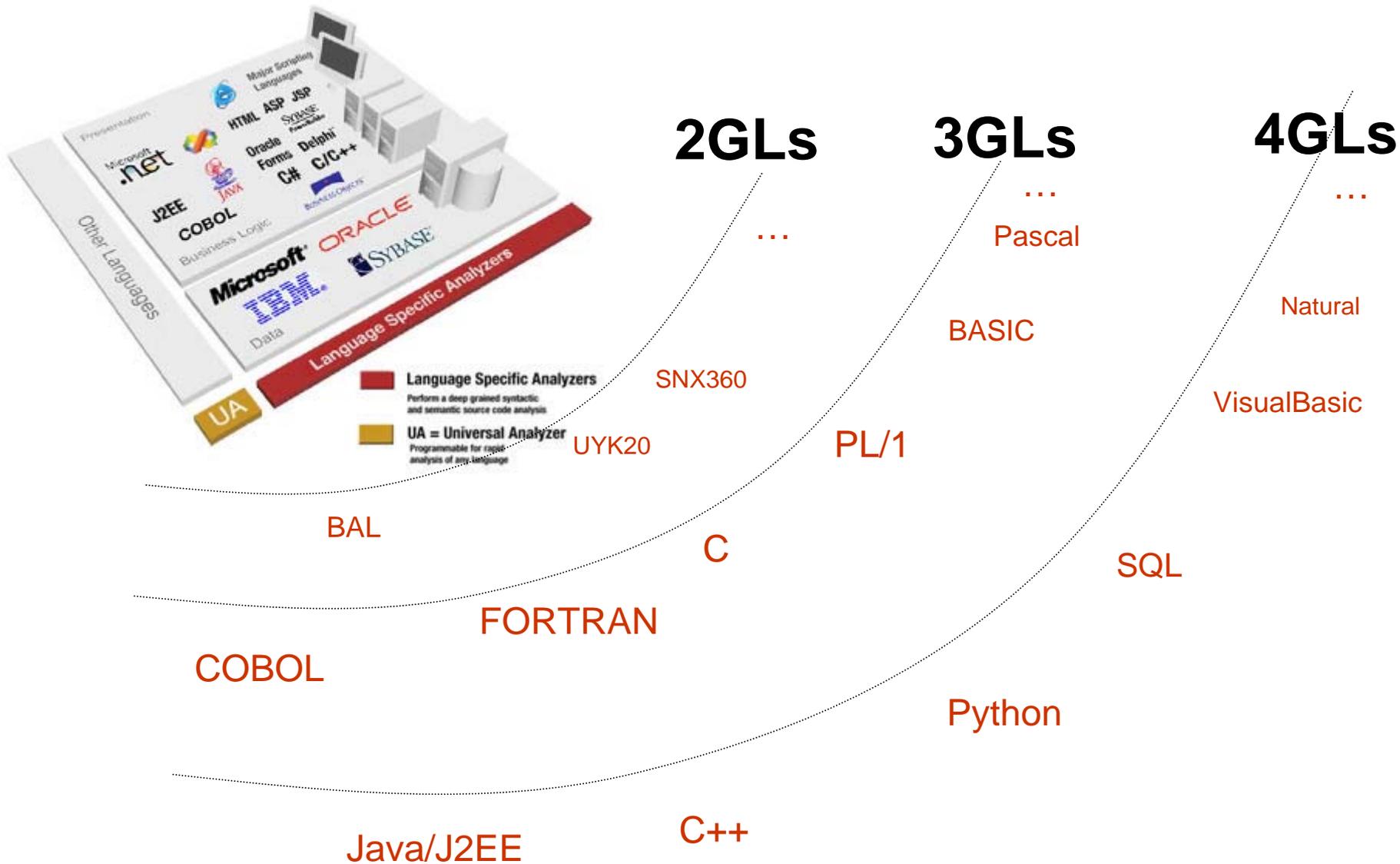
“In an era riddled with asymmetric cyber attacks, claims about system reliability, integrity and safety must also include provisions for built-in security of the enabling software.”

The Software Supply Chain

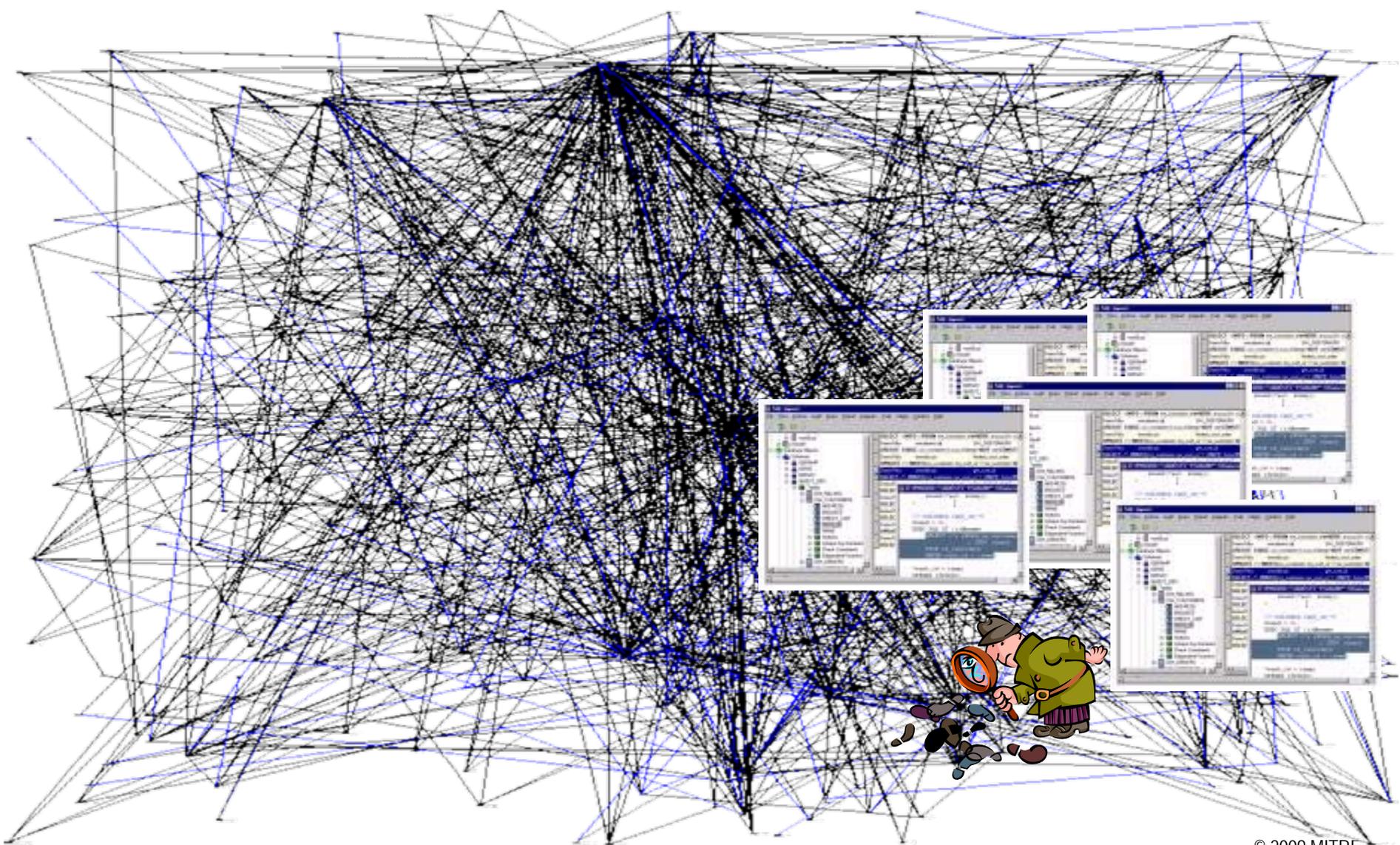


* "Scope of Supplier Expansion and Foreign Involvement" graphic in DACS www.softwaretechnews.com Secure Software Engineering, July 2005 article "Software Development Security: A Risk Management Perspective" synopsis of May 2004 GAO-04-678 report "Defense Acquisition: Knowledge of Software Suppliers Needed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

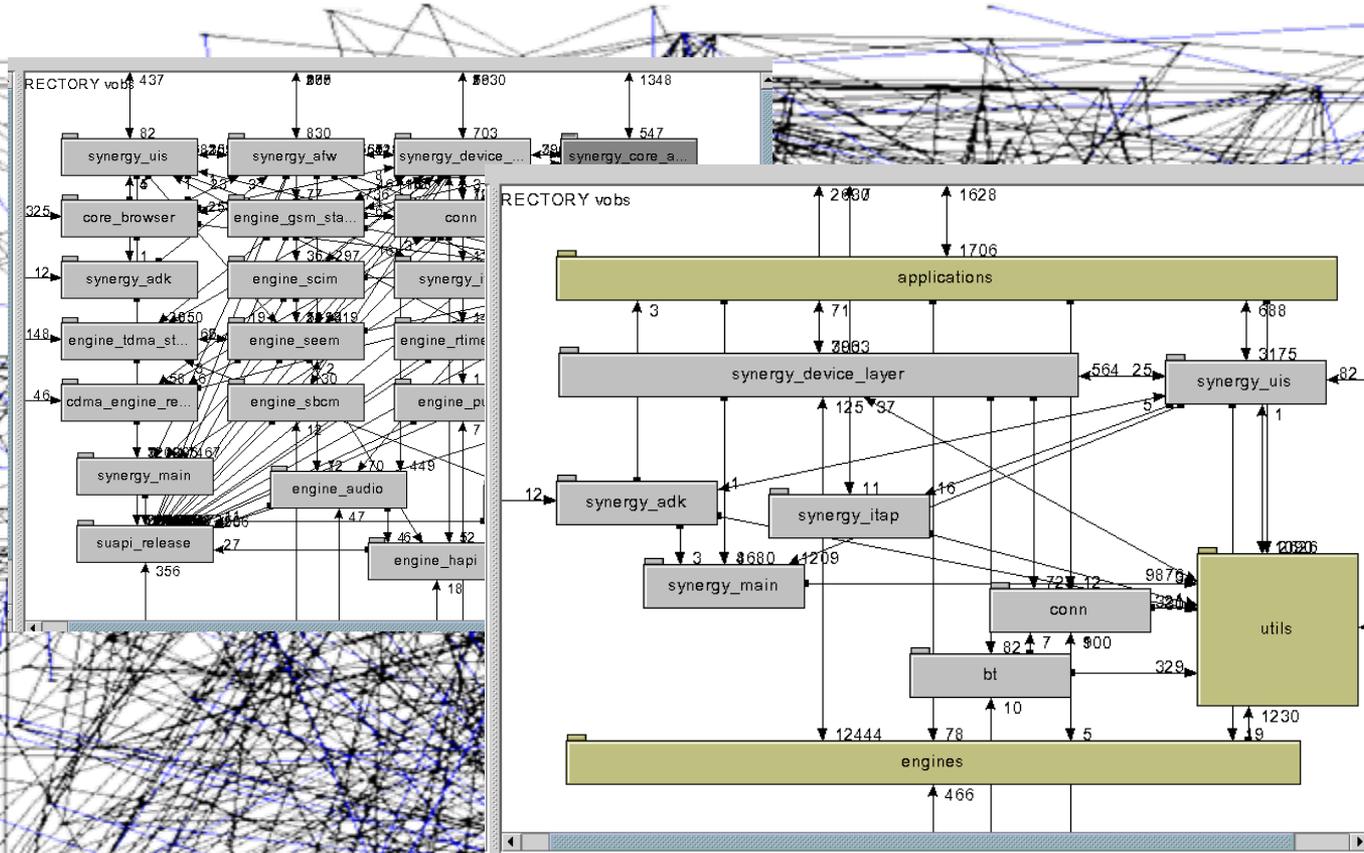
Our Systems are Composed of Elements from Many Languages and Environments



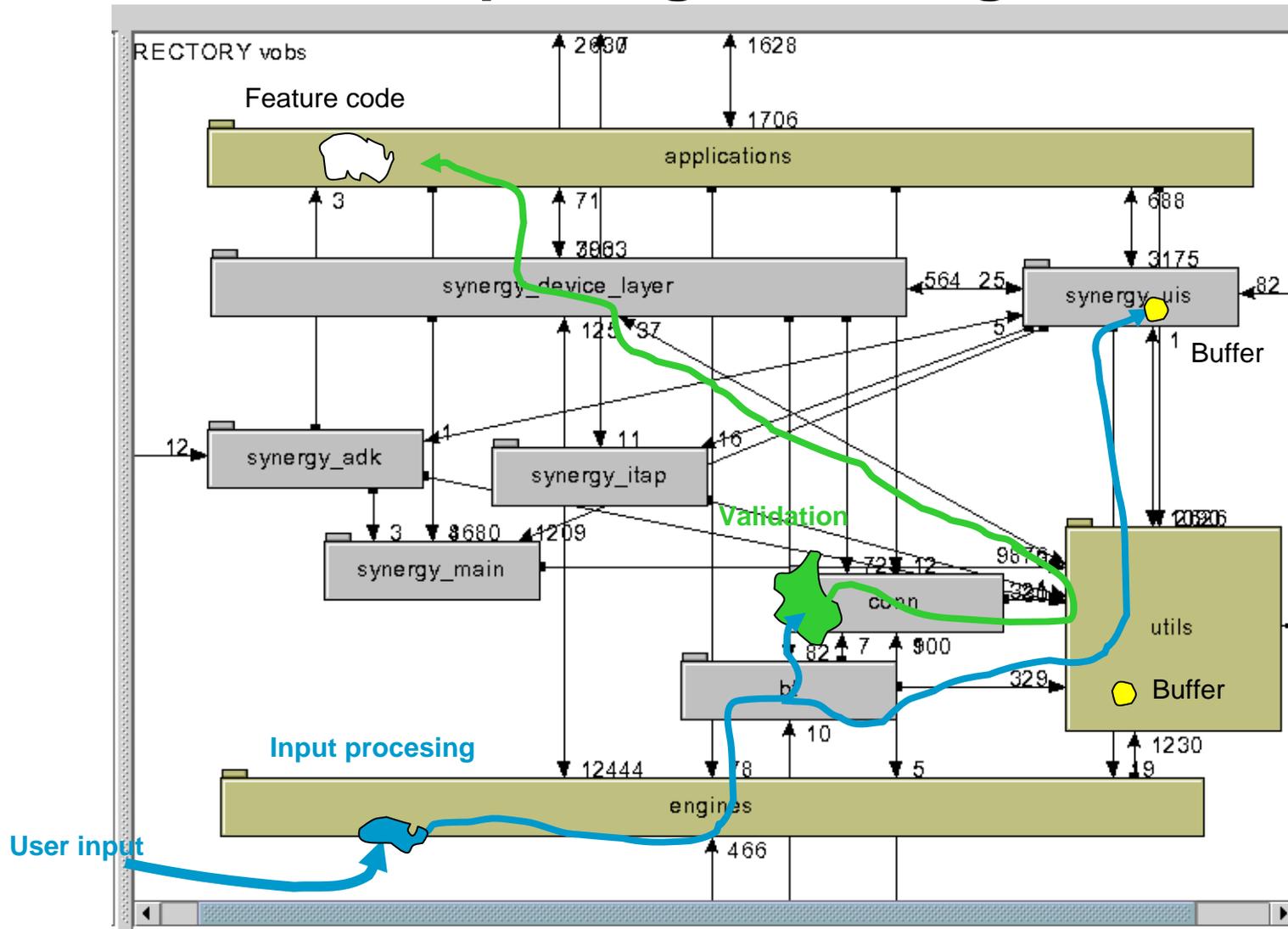
And Software Is Complex Too...



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



Static Analysis is about collecting information and capturing knowledge



Enterprise Applications



```
IDENTIFICATION DIVISION.
PROGRAM-ID.
AUTHOR.
COPY MEMBER1.
COPY MEMBER2.
```

Middleware

Web Services

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
AUTHOR.
COPY MEMBER1.
COPY MEMBER2.
```

CICS Connector

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
AUTHOR.
COPY MEMBER1.
COPY MEMBER2.
```

Batch Shell Scripts

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
AUTHOR.
COPY MEMBER1.
COPY MEMBER2.
```

Web/Client Server Applications

ASP/JSP/VB/.NET

```
<% page language="java" %>
<% page import="java.sql.*" %>
<% page import="com.castsoftware.
castpubs.jspbean.*" %>
<% page import="com.castsoftware.
framework.dbconnection.*" %>
<% page errorPage="ErrorPage.jsp" %>
```

Application Logic

Java, C++, ...
Frameworks Struts MVC, Spring

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
AUTHOR.
COPY MEMBER1.
COPY MEMBER2.
```

Data Management Layer

EJB - Hibernate - Jbatis

```
CREATE PROCEDURE dbo.EmployeesByStat
AS
BEGIN
SELECT e.LastName, e.Title, sp.
FROM HumanResource.Employee e
INNER JOIN Person.Contact c ON e
INNER JOIN HumanResource.Empl
```

Presentation Tier

Business Logic Tier

Data Tier

Legacy Applications

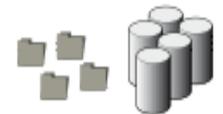
CICS:Monitor (Cobol)
Tuxedo Monitor (C)



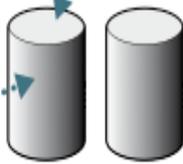
```
IDENTIFICATION DIVISION.
PROGRAM-ID.
AUTHOR.
COPY MEMBER1.
COPY MEMBER2.
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
AUTHOR.
COPY MEMBER1.
COPY MEMBER2.
```

COBOL



Files Databases

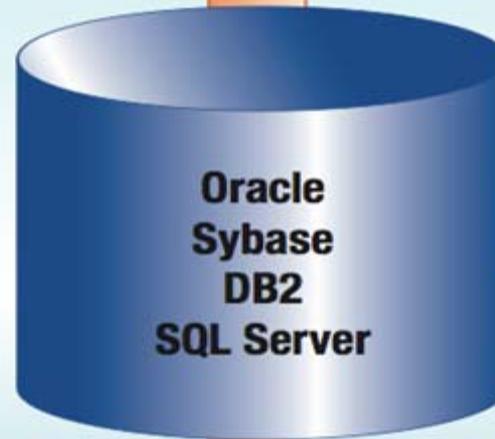


Database

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



Engines



SAMATE Reference Dataset

http://samate.nist.gov/SRD/

AFC Home MII Home Search Map/Ph/Weather/Travel Bob's Bookmarks CVerOVAL OVAL shared SPAMmngt

» sign in register | Search... GO



Software Assurance Metrics and Tool Evaluation

SRD Home View / Download Search / Download More Downloads Submit Test

Welcome to the NIST SAMATE Reference Dataset Project

The purpose of the SAMATE Reference Dataset (SRD) is to provide users, researchers, and developers with a 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 cycle (written to test or generated), and "academic" (from students) test cases. This dataset includes known bugs and vulnerabilities. The dataset intends to encompass a wide variety of test cases, compilers. The dataset is anticipated to become a large-scale effort, gathering test cases 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 here](#) to view 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

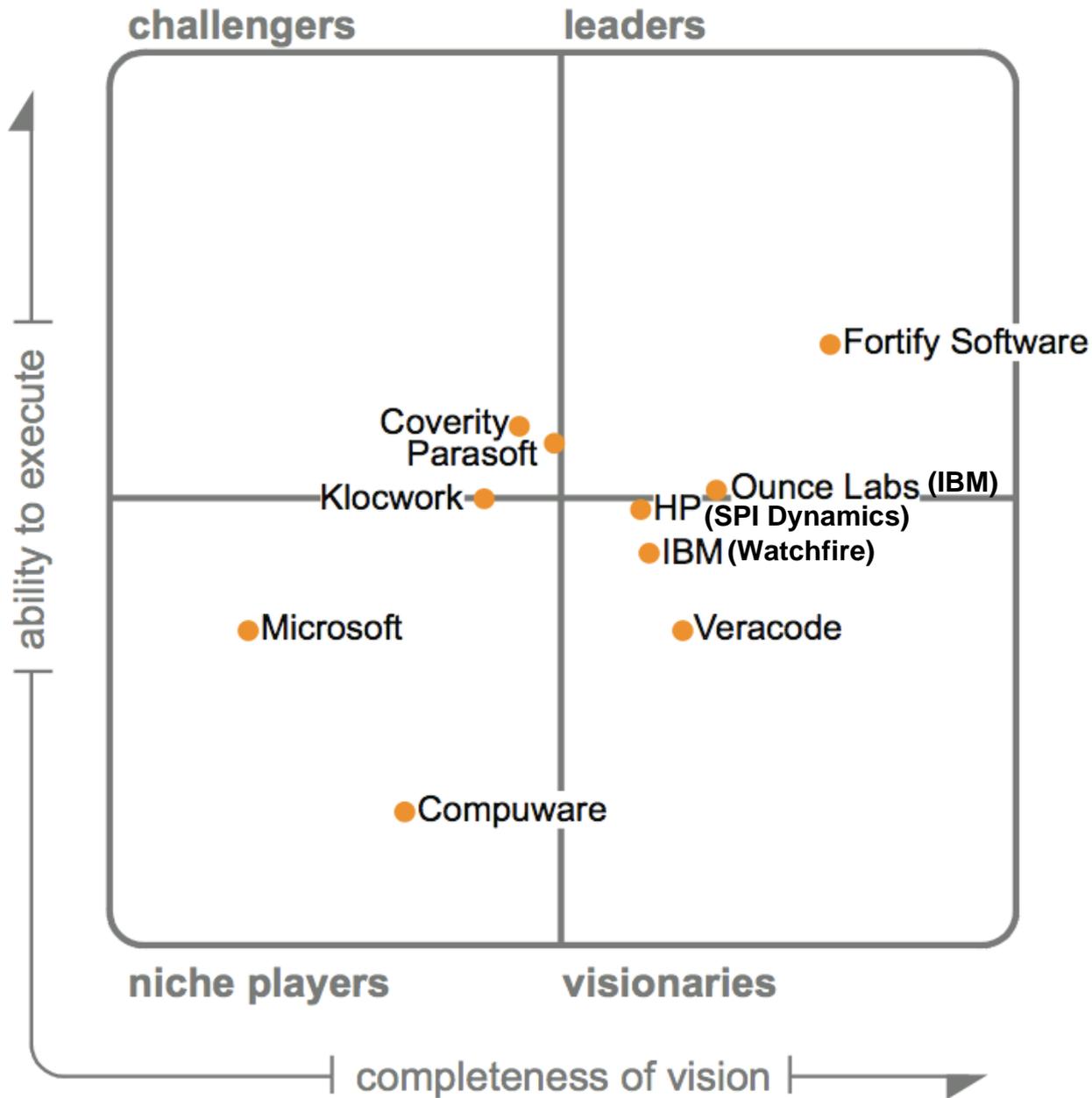
Information Technology Laboratory (ITL), Software
Diagnostics and Conformance Testing Division

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...

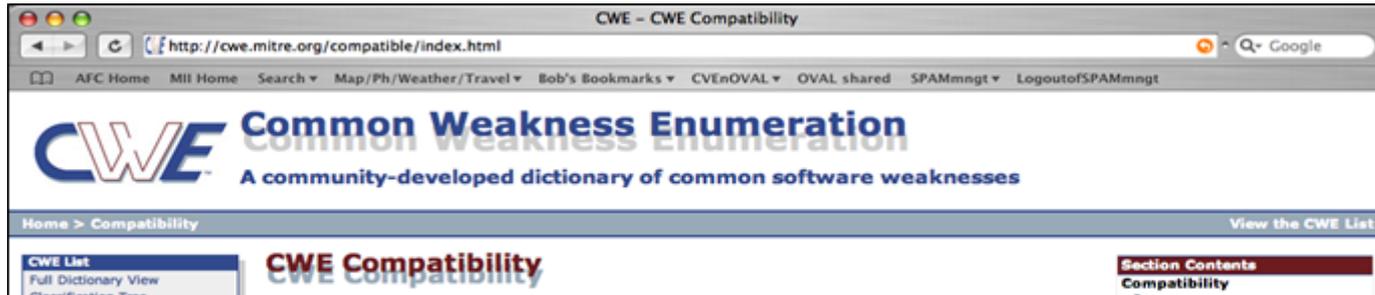
- Cenzic
- proServices
- Polyspace
- Security Innovation
- AppSIC Initiative
- KDAnalytics
- SureLogic
- Programming Research Inc
- Armorize
- Compuware
- SofCheck
- GrammaTech
- CORE Security

Source: Gartner

As of February 2009

CWE Compatibility & Effectiveness Program

(launched Feb 2007)



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.

Products are listed alphabetically by organization name:

cwe.mitre.org/compatible/

TOTALS
Organizations Participating: 28
Products & Services: 47

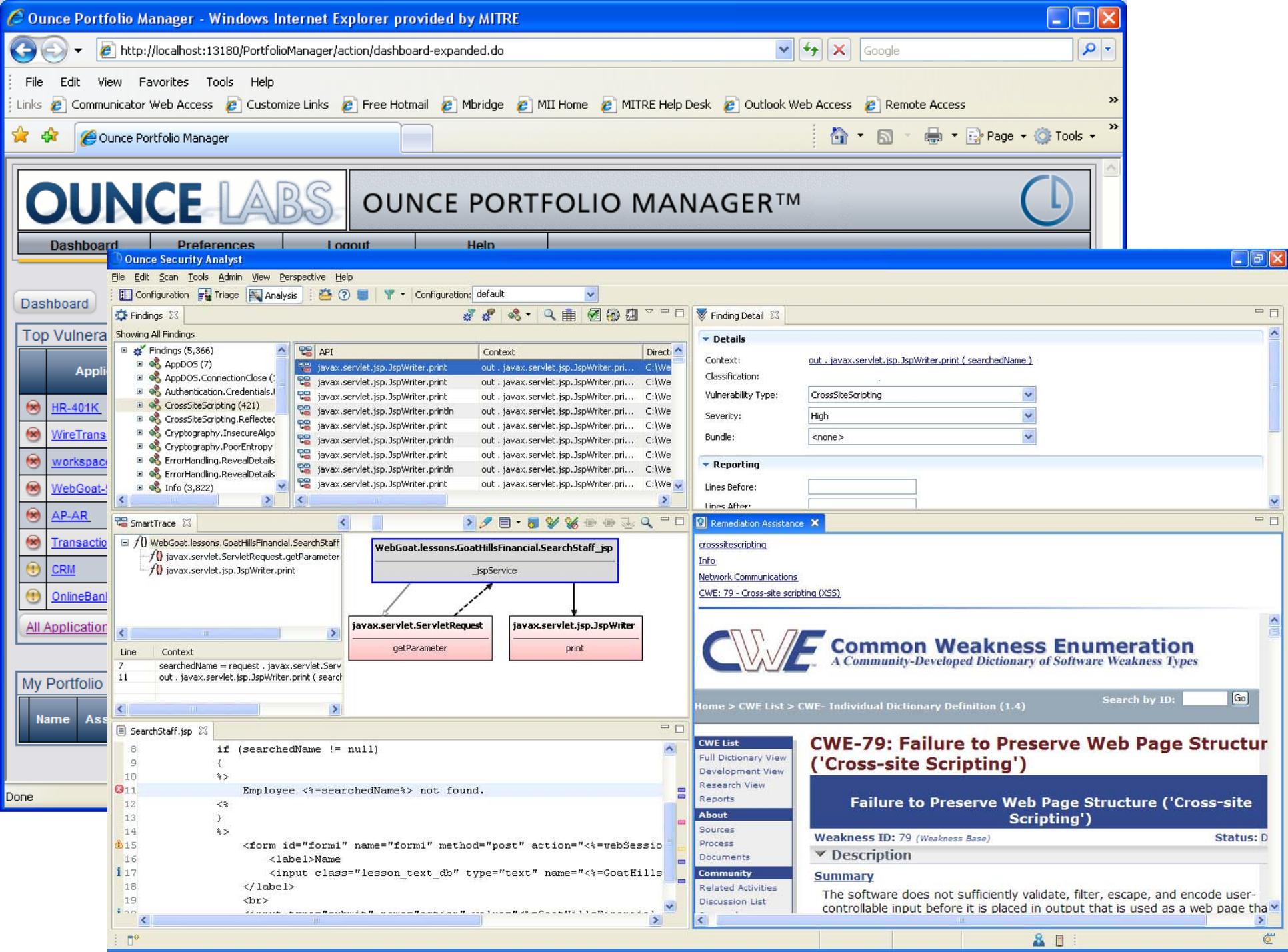
December 25, 2006

Fortify Main User Interface

“Eclipse” Look & Feel

The screenshot displays the Fortify Main User Interface, which has an Eclipse-like look and feel. The interface is divided into several panels:

- Flaw List w/menus to “slice and dice” the data:** Located in the top-left pane, it shows a list of flaws with various filters and sorting options.
- Source Code Pane:** The central pane displays the source code of the selected flaw, with a line of code highlighted.
- File & Directory Pane:** Located in the top-right pane, it shows a tree view of the project's file and directory structure.
- Stack trace variables, values, call trace:** Located in the bottom-left pane, it displays the stack trace for the selected flaw, including variable names, values, and call traces.
- Analysis mitigation suggestions, explanation of the flaw, mapping to CWE and some mappings to DISA Severity Codes:** The bottom-right pane displays detailed analysis results, including mitigation suggestions, an explanation of the flaw, and mappings to CWE and DISA Severity Codes.



Klocwork for C/C++ - client.c - Eclipse SDK

File Edit Navigate Search Project Run Window Help

Navigator

```

2165 /* Save this file to retrieve later.
2166 failed_patches = (char **) xrealloc
2167     ((failed_patches_count
2168      * sizeof (char **))
2169      failed_patches[failed_patches_count]
2170      ++failed_patches_count;
2171
2172 stored_checksum_valid = 0;
2173
2174 free (mode_string);
2175 free (buf);
2176 free (scratch_entries);
2177 free (entries_line);
2178
2179 return;
2180
2181
2182 {
2183     int status = change_mode (filename, r
2184     if (status != 0)
  
```

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

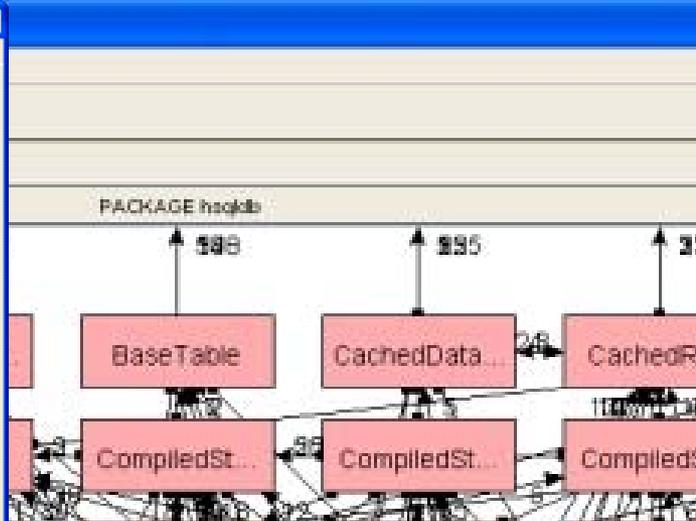
- client.c:1806: data->contents==UPDATE_ENTRIES_
- client.c:1849: data->existp==UPDATE_ENTRIES_
- client.c:2027: data->contents==UPDATE_ENTRIES_
- client.c:2031: data->contents==UPDATE_ENTRIES_
- client.c:2066: !rcs_change_text(short_pathname
- <library>: in rcs_change_text
- client.c:2116: stored_checksum_valid&& !patch_f
- client.c:2163: patch_failed is true
- client.c:2179: Dynamic memory stored in 'patched

Klocwork Findings

Klocwork Log Console

Visible 226 of 226 items. Grouped by None. Sorted by Severity, then by Description

Description	Resource	Loca...
SV XSS DEF: Detecting user input()...



Klocwork Insight Review - Mozilla Firefox

http://klocwork.com/8070/klocwork/central.html;sessionid=A37585A39D15CBF85D58556AD3834F3#historyID_b2UD21e3l

Klocwork Insight Review

Project List Reports Issue Management Source Cross-Reference About Help

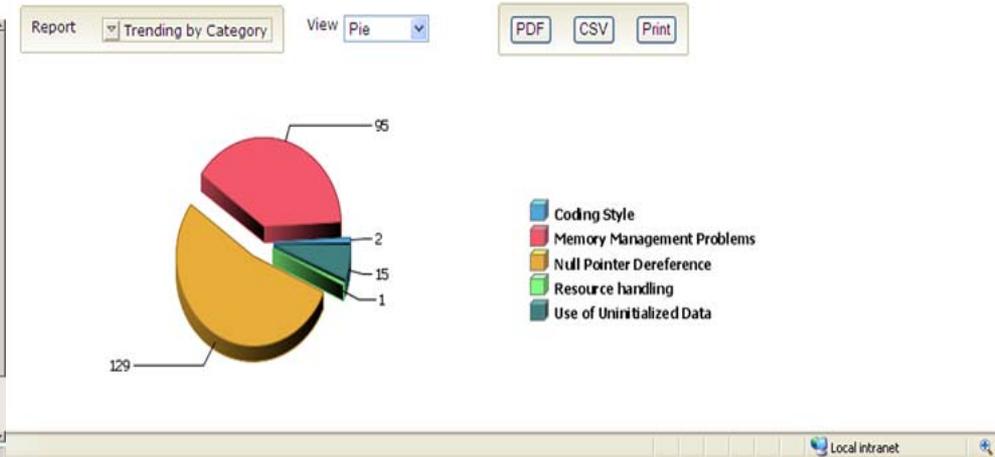
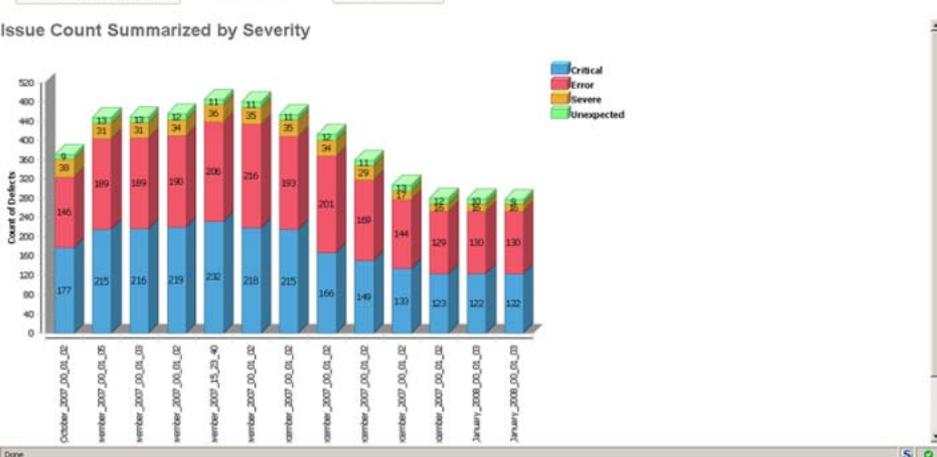
Search by: Criteria Issue ID Saved Scopes: *unsaved* Live...

State	New	Existing	Not in Scope	Fixed	Recurred	none					
Severity	Critical	Severe	Error	Unexpected	Investigate	Warning	Suggestion	Style	Review	Info	none
Status	Analyze	Ignore	Not a Problem	Fix	Fix in Next Release	Fix in Later Release	Defer	Filter	none	none	

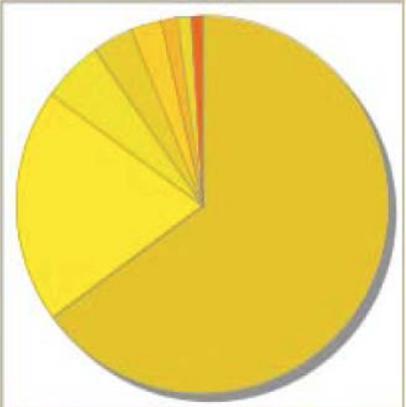
Builds: build_1

Components: all components

Categories: 15 items: Coding Style, Concurrency, Memory Management, Array Index Out of Bounds, Unvalidated User Input Ca, Null Terminated String, Use of Freed Memory on Return.

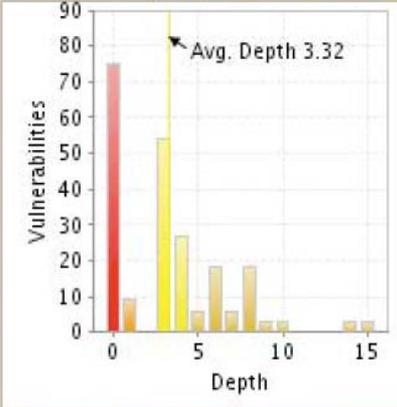


Vulnerability Type Distribution



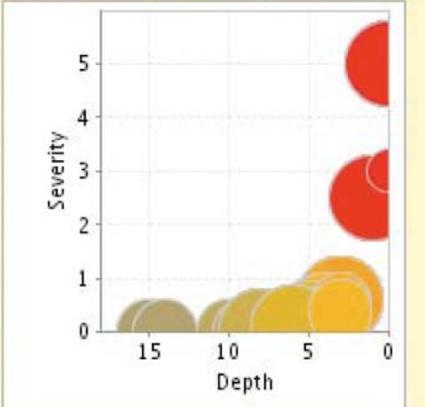
- Cross-Site Scripting (CWE 79)
- SQL Injection (CWE 89)
- Resource Injection (CWE 99)
- HTTP Response Splitting (CWE 113)
- Command Injection (CWE 77)
- Hard-Coded Password (CWE 259)
- XPath Injection (CWE 91)
- Information Leak of System Data (CWE 497)

Vulnerability Depth Distribution



- Directly exposed
- Vulnerability
- Vulnerability
- Vulnerability

Vulnerability Severity Distribution



Multi-Language Support

- J2EE (Java, JSP)
- .NET
- ASP
- PHP

Vulnerability List

File Name	Vulnerability Type
DefaultLessonAction.java	Information Leak of System Data (CWE 497)
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

```

<LINK href="include/style.css" rel="stylesheet">
</link>
<body>
<table width="95%" border="0" cellpadding="0" cellspacing="1" align="center" height="51">
<tr>
<td align="center" height="17"><div class="green"><div style="float: right; text-align: right; font-size: 12px; font-weight: bold; color: #000080; text-decoration: underline; padding: 2px 0 2px 10px;">
</div>
</td>
<td align="center" height="19"><a href="#" onclick="window.close()"> <small style="font-size: 10px; color: #000080; text-decoration: underline;">Close Window</a></td>
</tr>
</table>
<table width="95%" border="0" cellpadding="0" cellspacing="0" class="tablecss" align="center">
<tr>
<td width="70%"><div style="float: right; text-align: right; font-size: 12px; font-weight: bold; color: #000080; text-decoration: underline; padding: 2px 0 2px 10px;">
</div>
</td>
<td align="right"><a href="#"> <small style="font-size: 10px; color: #000080; text-decoration: underline;">Close Window</a></td>
</tr>
</table>
<tr>
<td align="center" height="17"><div class="green">
</td>
</tr>
<tr>
<td align="center" height="19"><a href="#"> <small style="font-size: 10px; color: #000080; text-decoration: underline;">Close Window</a></td>
</tr>
</table>

```

Scan Result
Report
Console

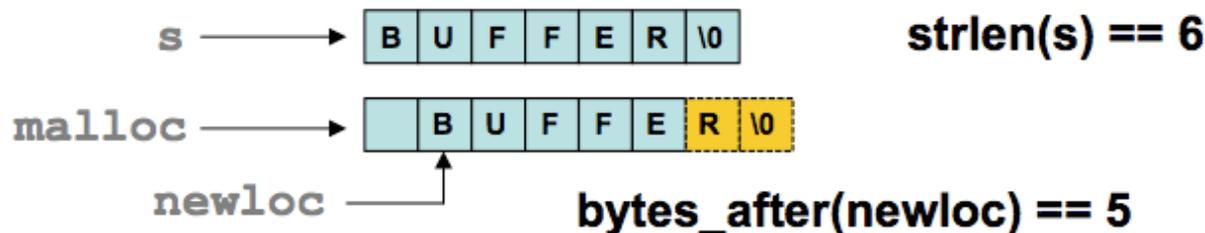
/: ublog - 11:46:34 AM

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_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
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	<code>char *return_append_str(char *dest, const char *s) {</code>
	1767	<code>/* Append text s to dest, and return new result. */</code>
	1768	<code>char *newloc;</code>
	1769	<code>size_t newlen;</code>
	1770	<code>/* This doesn't have buffer overflow vulnerabilities, because</code>
	1771	<code>we always allocate for enough space before appending. */</code>
	1772	<code>if (!dest) {</code>
true	1773	<code>newloc = (char *) malloc(strlen(s)+1;</code>
strlen(s) > bytes_after(newloc) - 1	1774	<code>strcpy(newloc, s); /* Buffer Overrun */</code>
+ Preconditions		
+ Postconditions		



Example Buffer Overflow

Package:

File(Line): ExternalloModel/ExternalloPkg/.../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], p
2213 if(crcCalc != crcRcv)
2214 {
```

```
EioJreapDecoder.cpp:403 - Caller: decodeMGH
Buffer Size: 2
EioJreapDecoder.cpp:403 - Caller: decodeMGH
Buffer Size: 2
EioJreapDecoder.cpp:2203 - memcpy()
Buffer Size: 9207 bytes
Write Length: 9216 bytes
bytesNeeded: 9216
```

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 //#[ of
189 CINT F
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
resentation, 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
 - 1. Duplicates the buffer into a temp buffer → duplicates **sprintf()** call poorly
 - 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  
1344 VOID 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::Q  
1351     if(queueIndex >= 0)  
1352     {  
1353         m_FwdRcvMgr->receive(a)omIndex(getMbufP(), queueIndex, b)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

- The `m_OwnUnitIcmSize` value is looked up via `getMbufP().getCriteria(EIO_CRITERIA_ICM_SIZE)`
- If the (unsigned integer) `m_OwnUnitIcmSize` is less than, or equal to (integer), `EIO_JREAP_PPLI_DATA_SIZE`
- Then copy that `m_OwnUnitIcmSize` number of bytes from the pointer returned by the `getMbufP().getMbufPtr()` call into the `m_OwnUnitIcmBufferData` memory location.
- Large, positive integer values will flip the highest bit. A signed comparison would consider that value to be negative.
- 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](#)

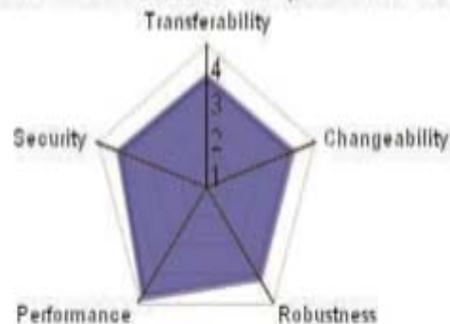
Quality and Quantity

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

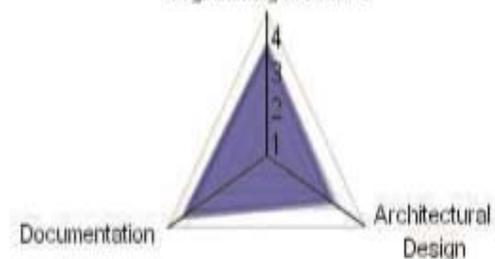


Maintainability (SEI)	3.43	3.43
---------------------------------------	-------------	-------------

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

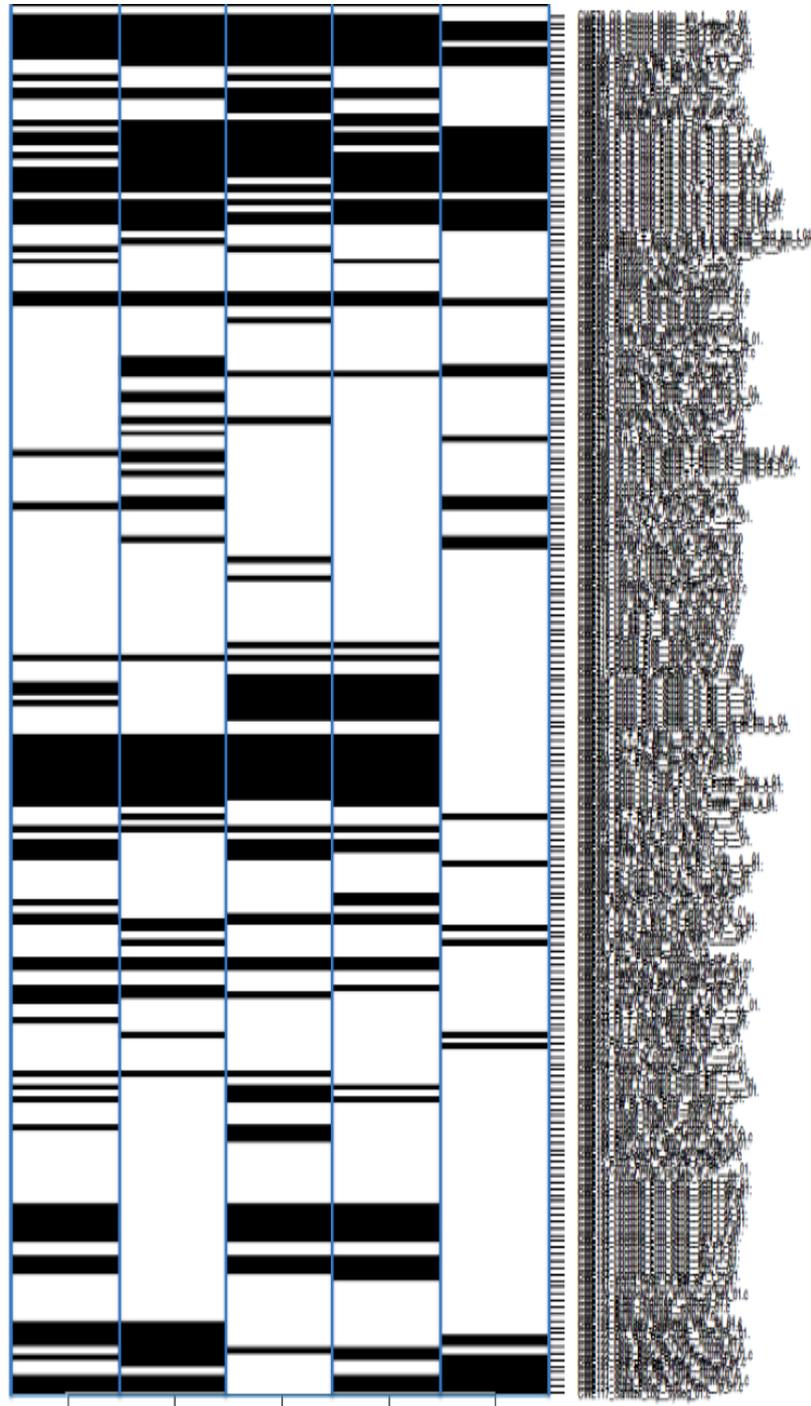
Rule Compliance of Application Legacy HR application in Apr '06 snapshot

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 -

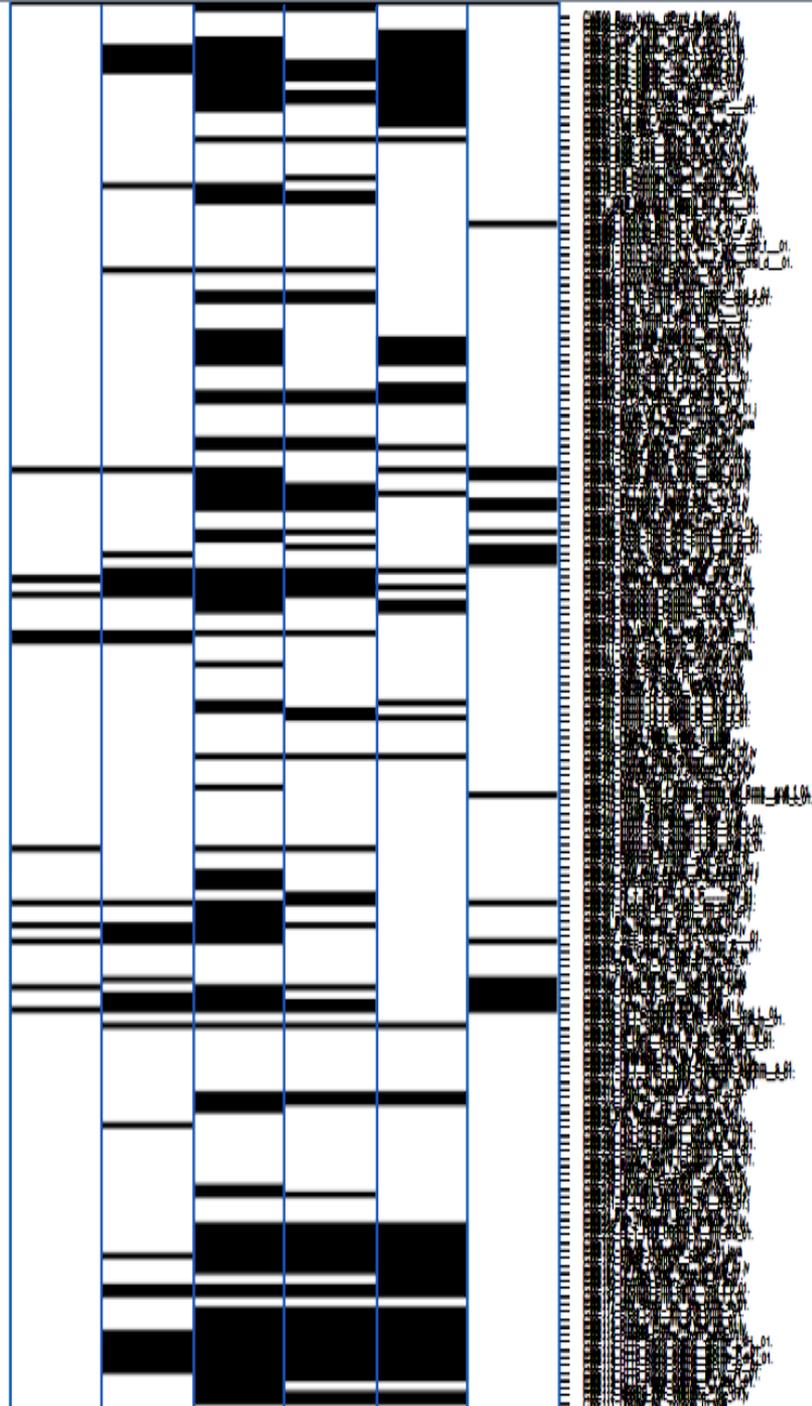


Programming Practices	2.97
Architectural Design	2.6
Documentation	3.34

C Test Cases

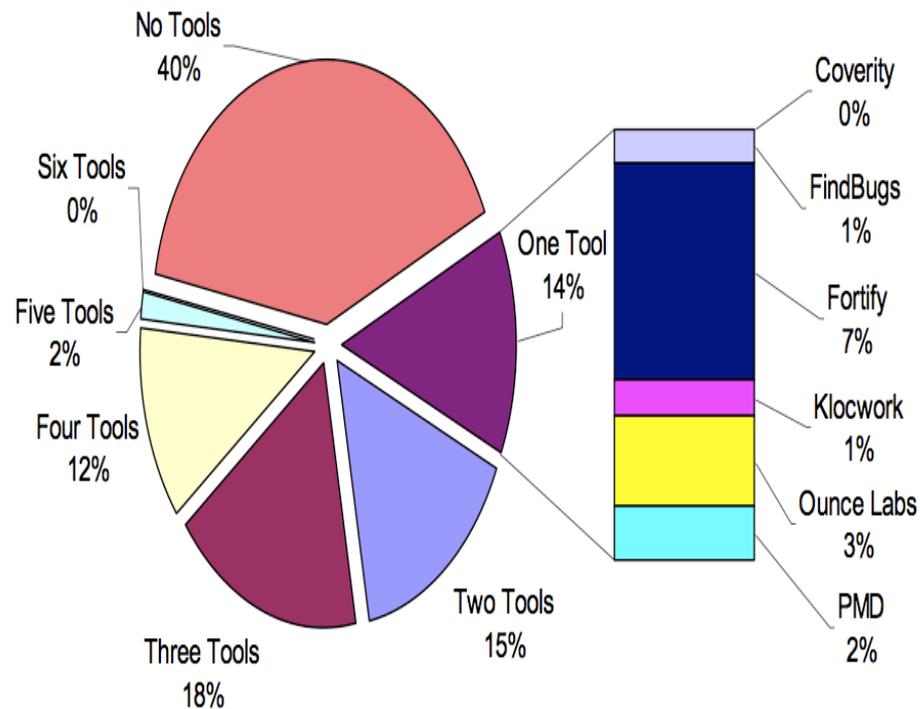
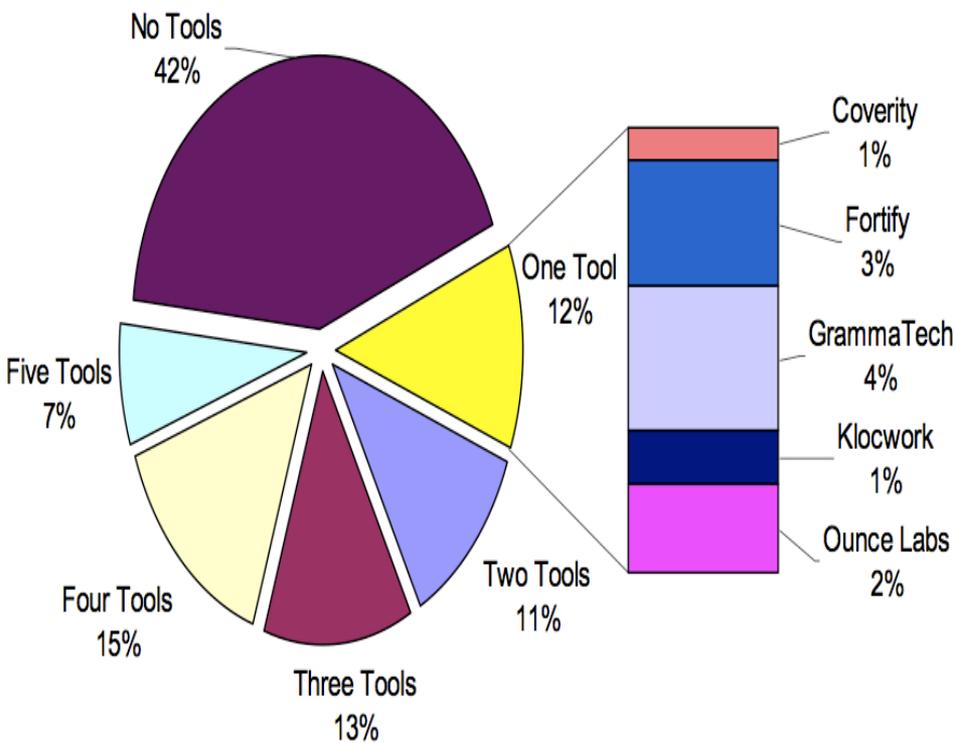


Java Test Cases



C/C++ "Breadth" Test Case Coverage

Java "Breadth" Test Case Coverage





Attack Pattern 7
ID

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

Individual CAPEC Dictionary Definition (Release 3.2)

Stance: Required, Incomplete

Attack Pattern ID: A10

Attack Pattern Name: Denial of Service

Attack Pattern Description: A denial of service attack is an attack that aims to make a computer system or network unavailable to its intended users. This is typically achieved by flooding the target with a large volume of traffic, or by exploiting a vulnerability in the target's software to cause it to crash or become unresponsive.

Attack Pattern Type: Denial of Service

Attack Pattern Status: Active

Attack Pattern Category: Availability

Attack Pattern Subcategory: Denial of Service

Attack Pattern Mitigation: Implement network security measures, such as firewalls and intrusion detection systems, to prevent unauthorized access to the network. Regularly update software and hardware to address known vulnerabilities.

Attack Pattern Reference: [https://www.mitre.org/attack-patterns/A10](#)

Attack Pattern ID: A10

Attack Pattern Name: Denial of Service

Attack Pattern Description: A denial of service attack is an attack that aims to make a computer system or network unavailable to its intended users. This is typically achieved by flooding the target with a large volume of traffic, or by exploiting a vulnerability in the target's software to cause it to crash or become unresponsive.

Attack Pattern Type: Denial of Service

Attack Pattern Status: Active

Attack Pattern Category: Availability

Attack Pattern Subcategory: Denial of Service

Attack Pattern Mitigation: Implement network security measures, such as firewalls and intrusion detection systems, to prevent unauthorized access to the network. Regularly update software and hardware to address known vulnerabilities.

Attack Pattern Reference: [https://www.mitre.org/attack-patterns/A10](#)

Stub's Information

Attack Pattern ID: A10

Attack Pattern Name: Denial of Service

Attack Pattern Description: A denial of service attack is an attack that aims to make a computer system or network unavailable to its intended users. This is typically achieved by flooding the target with a large volume of traffic, or by exploiting a vulnerability in the target's software to cause it to crash or become unresponsive.

Attack Pattern Type: Denial of Service

Attack Pattern Status: Active

Attack Pattern Category: Availability

Attack Pattern Subcategory: Denial of Service

Attack Pattern Mitigation: Implement network security measures, such as firewalls and intrusion detection systems, to prevent unauthorized access to the network. Regularly update software and hardware to address known vulnerabilities.

Attack Pattern Reference: [https://www.mitre.org/attack-patterns/A10](#)

CAPEC Current Content (12 Major Categories)

1000 - Mechanism of Attack

Data Leakage Attacks - (118)

Resource Depletion - (119)

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

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 structure - (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...)



Threat Landscape

- Complex software and hardware have many vulnerabilities
- Many critical components developed by foreigners
- IT supply
- Sophistic
- increasing



Cyber Threats

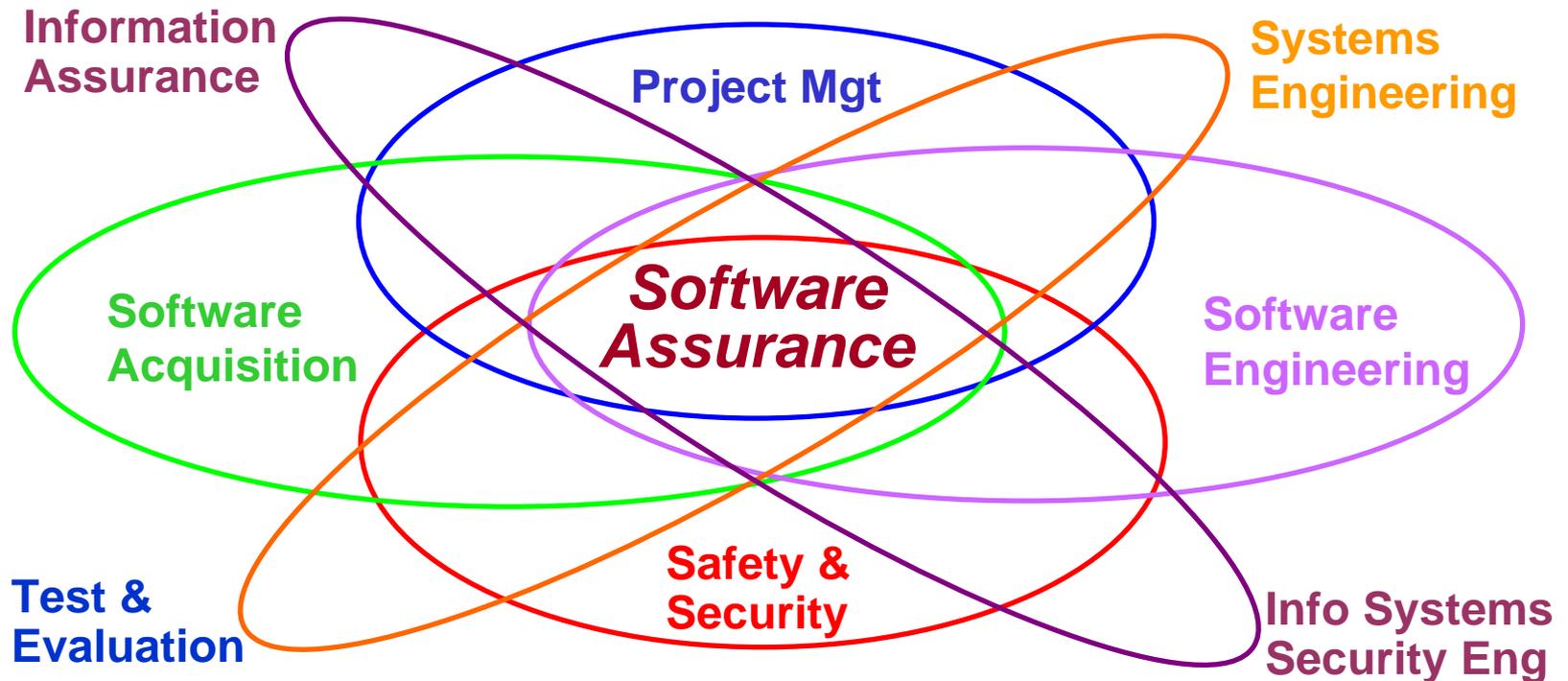
- “All” software “can” have vulnerabilities
- Critical software can have “un-vetted” creators
- Many places to “attack” the IT supply chain



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](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

- λ **Software vulnerabilities jeopardize infrastructure operations, business operations & services, intellectual property, and consumer trust**
- λ **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**
- λ **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”**
- λ **Processes and technologies are required to build trust into software**



**Homeland
Security**

Strengthen operational resiliency



DoD Perspective on the Software Assurance (SwA) Problem



- λ Software is critical to the Global Information Grid, most weapons, business and support systems
- λ 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**
- λ 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*

- λ Software is the core constituent of modern products and services – it enables functionality and business operations
- λ 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**
- λ 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

- λ Systems are at risk due to software content & threat environment
- λ Software assurance is a significant part of Mission Assurance
- λ Significant risks come from
 - **Human coding mistakes and design flaws leading to security flaws**
 - **Supply Chain compromises**
- λ 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 threat information about IT suppliers

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

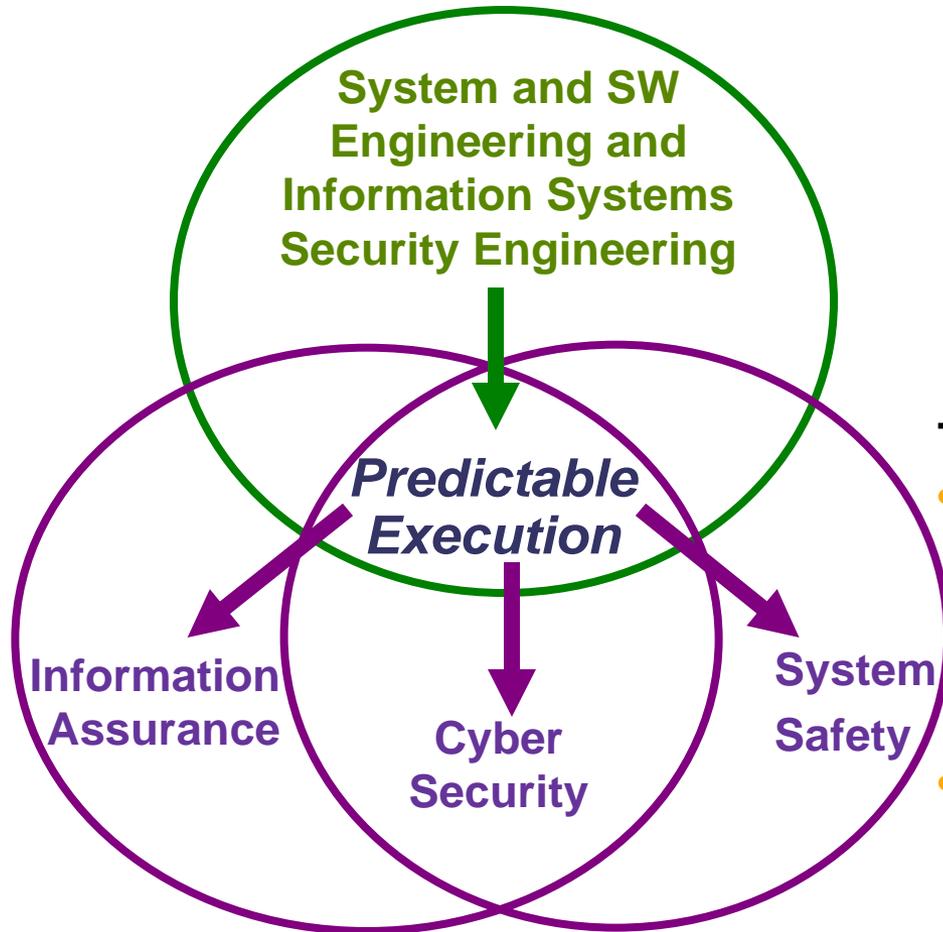
Find weaknesses in:

- 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

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

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



Understanding what we built / acquired

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



Using what we understand

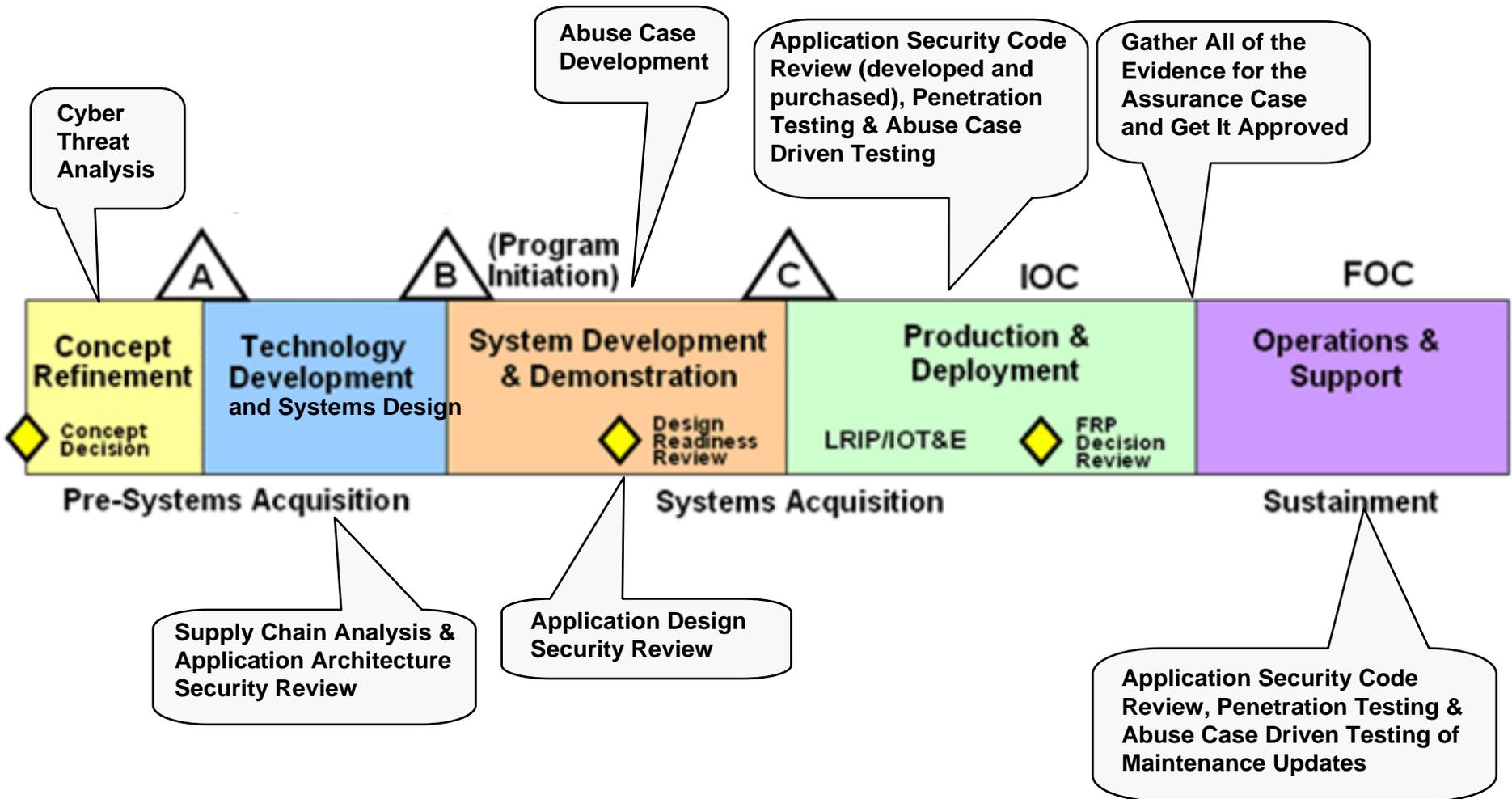
- ▶ Policy/practices for use & acquisition
- ▶ Composition of trust
- ▶ Hardware support



*Multiple Sources:

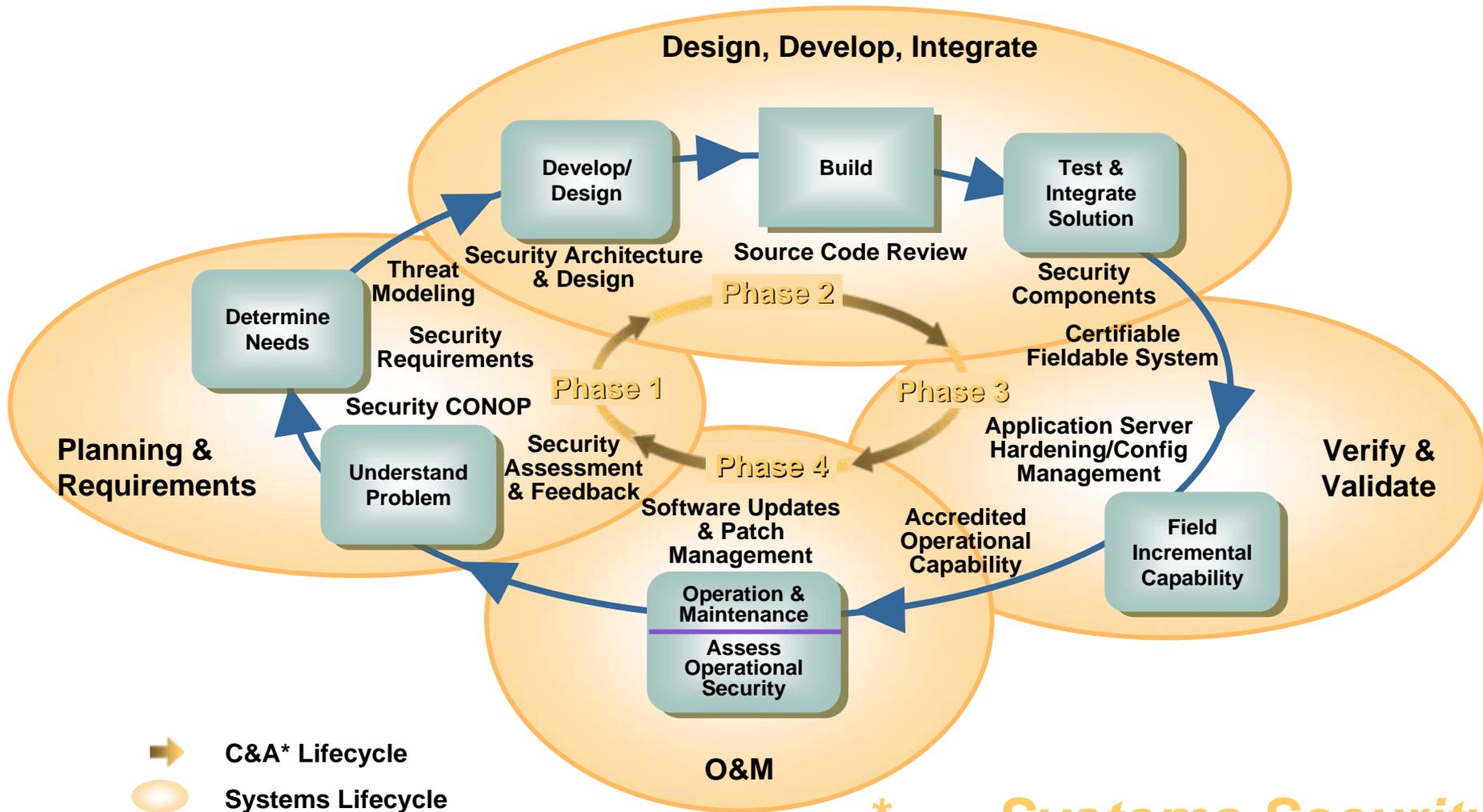
DHS/NCSA, OASD(NII)IA,
NSA, NASA, JHU/APL

SwA and Systems Development (example)



* Ideally Insert SwA before RFP release

Integrating SwA into the Systems Engineering Lifecycle



-  C&A* Lifecycle
-  Systems Lifecycle
-  Security Lifecycle

Software Assurance Lifecycle Considerations

- λ **Define Lifecycle Threats/Hazards, Vulnerabilities & Risks**
- λ **Identify Risks attributable to software**
- λ **Determine Threats (and Hazards)**
- λ **Understand key aspects of Vulnerabilities**
- λ **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: Policies & Practices, Constraints**

The Assurance Case/Argument – Requires Measurement

- λ 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**
- λ 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**


[About SwAC](#)
[Projects](#)
[Resources](#)
[Events](#)
[Members](#)
[Contact Us](#)
[Home > Projects](#)
[Software Facts](#)
[Similar Programs](#)
[Scope](#)
[Content & Criteria](#)
[Participants](#)
[Software Package Label](#)

Software Facts

Wouldn't it be great if software came with labels like food does? In 2004 [Aspect Security](#) proposed having a set of software facts, similar to a [nutrition facts label](#), [material safety data sheets](#), or [laser safety classes](#). Like food, it would not tell you everything about the software, but could give you some ideas about its content. It would be a step toward making the asymmetrical flow of information (see George Akerlof, "The Market for Lemons" 1970) more symmetrical and might lead to markets for better (pick your definition of "better") software.

The [Software Assurance Consortium](#) (SwAC) is the home for this software facts effort and takes on this activity as a special SwAC project.

Cautions

A fixed, small collection of software facts could be harmful. Here are some cautions.

- A label can give false confidence. It may suggest security when there is not enough. ("Fat free? Great, I'll have six!")
- A cursory review of the label may be done instead of appropriate analysis of other, existing material.
- A label may become de rigueur and shut out better software.
- A label could entrench current art and thus slow progress, either research or adoption.
- Lack of an attribute in the label may lead to the assumption that the attribute is missing
- Earning the label might divert effort from **real** product improvements.
- Developing a label might divert effort better used to directly research software assurance. (Don't bother standardizing buggy whips)
- A label may duplicate existing statutes, bargaining rights, or due diligence.
- A label can lead to liability and prosecution - **not** what we want.
- It could be outdated by patches or a new version OR be too bothersome to get for each new micro-version.

The basic idea here is expounded from Aspect Security's ideas. I am grateful to them, particularly Jeff Williams, for sharing their ideas.

The following web pages have additional information:

- [Similar programs](#) and related efforts
- Possible [scope](#), including audiences, classes of products or services, goals, and terminology.
- Possible [content](#) and criteria for content
- [Participants](#) and eventually process issues
- A proposed [software package label](#)

Next Steps

To learn more about this effort or to get involved, please contact Daniel G. Wolf (dewolf@SwACconsortium.org) at the [Software Assurance Consortium](#) or Paul E. Black (paul.black@nist.gov) at the U.S. National Institute of Standards and Technology (NIST).

Next steps are to organize a committee or group and start narrowing down the process, scope, and content.

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%

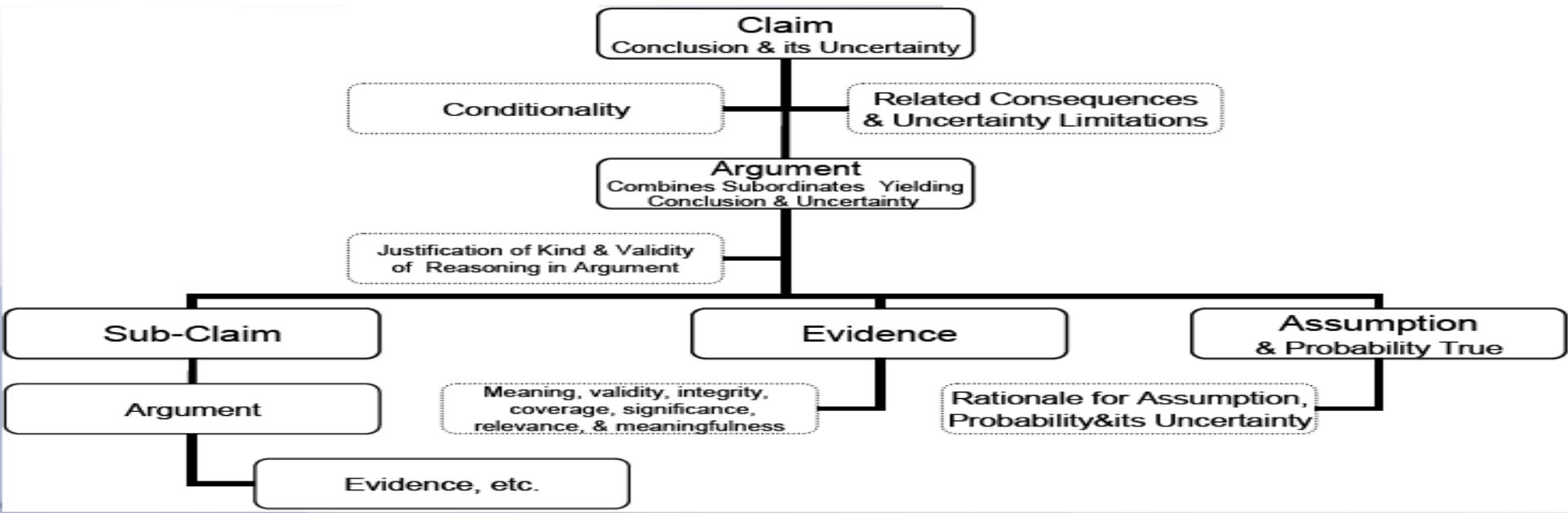
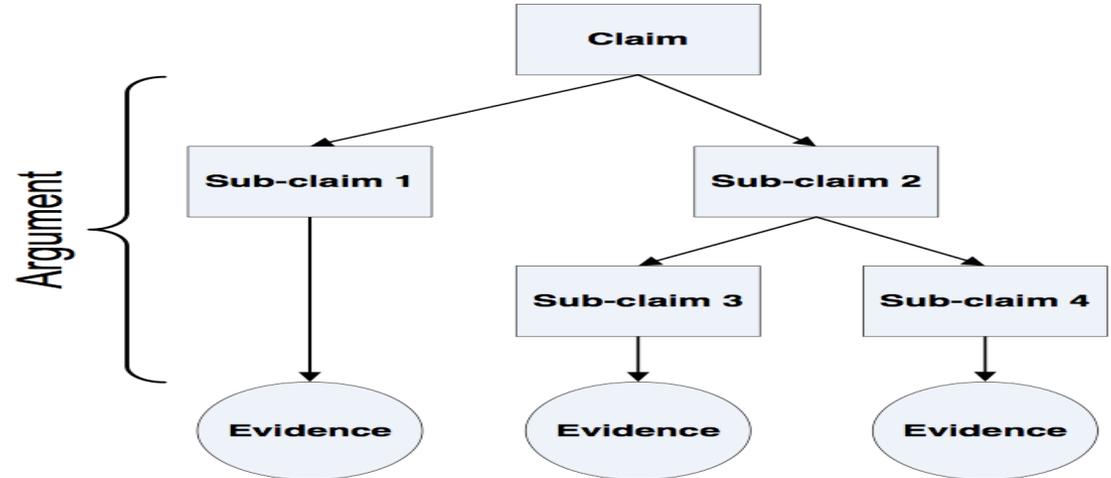
Libraries: Sun Java 1.5 runtime, Sun J2EE 1.2.2, Jakarta log4j 1.5, Jakarta Commons 2.1, Jakarta Struts 2.0, Harold XOM 1.1rc4, Hunter JDOMv1

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

λ 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)

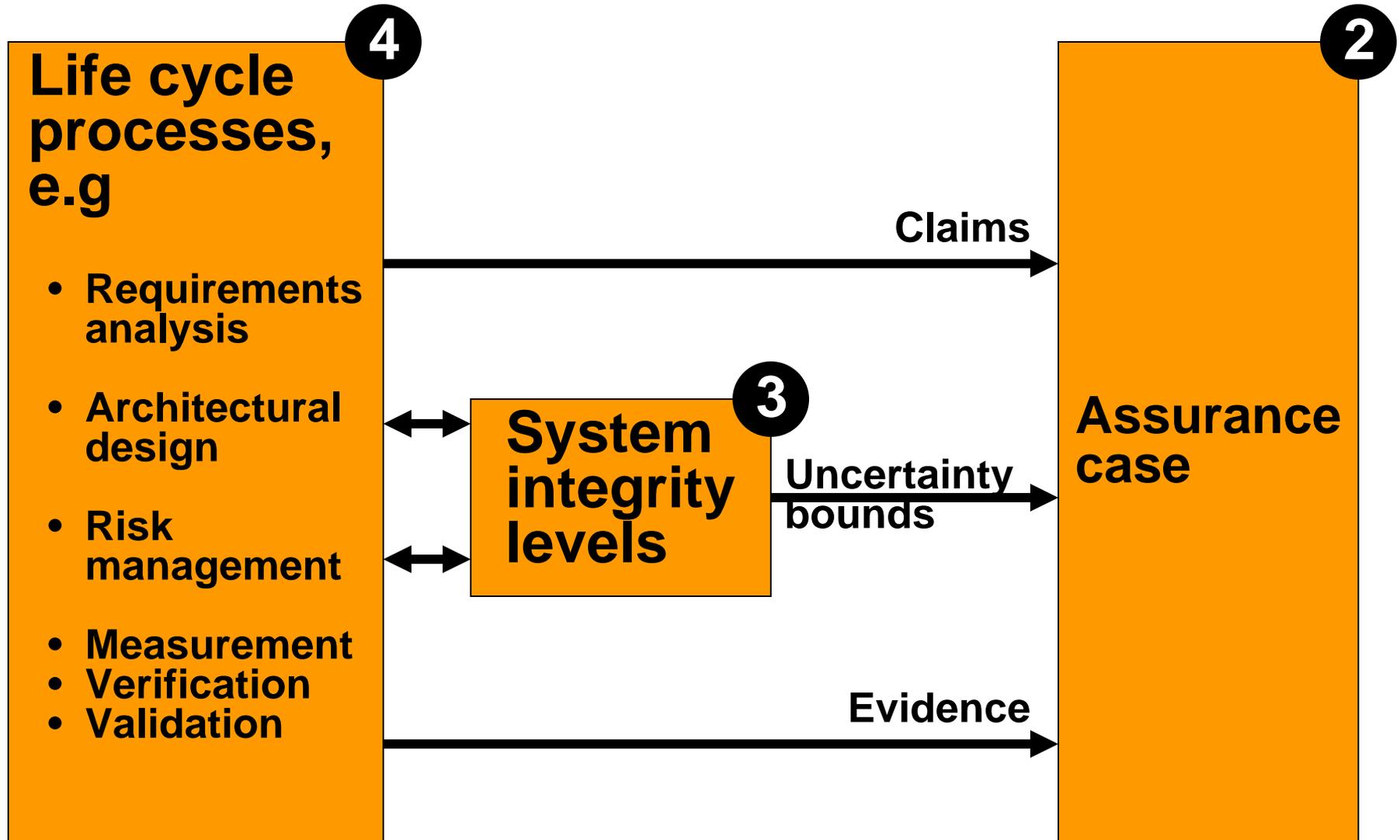
15026-4: Assurance in the life cycle (including project planning for assurance considerations)

λ Possible additional parts as demand requires and resources permit, e.g.

Assurance analyses and techniques

Guidance documents

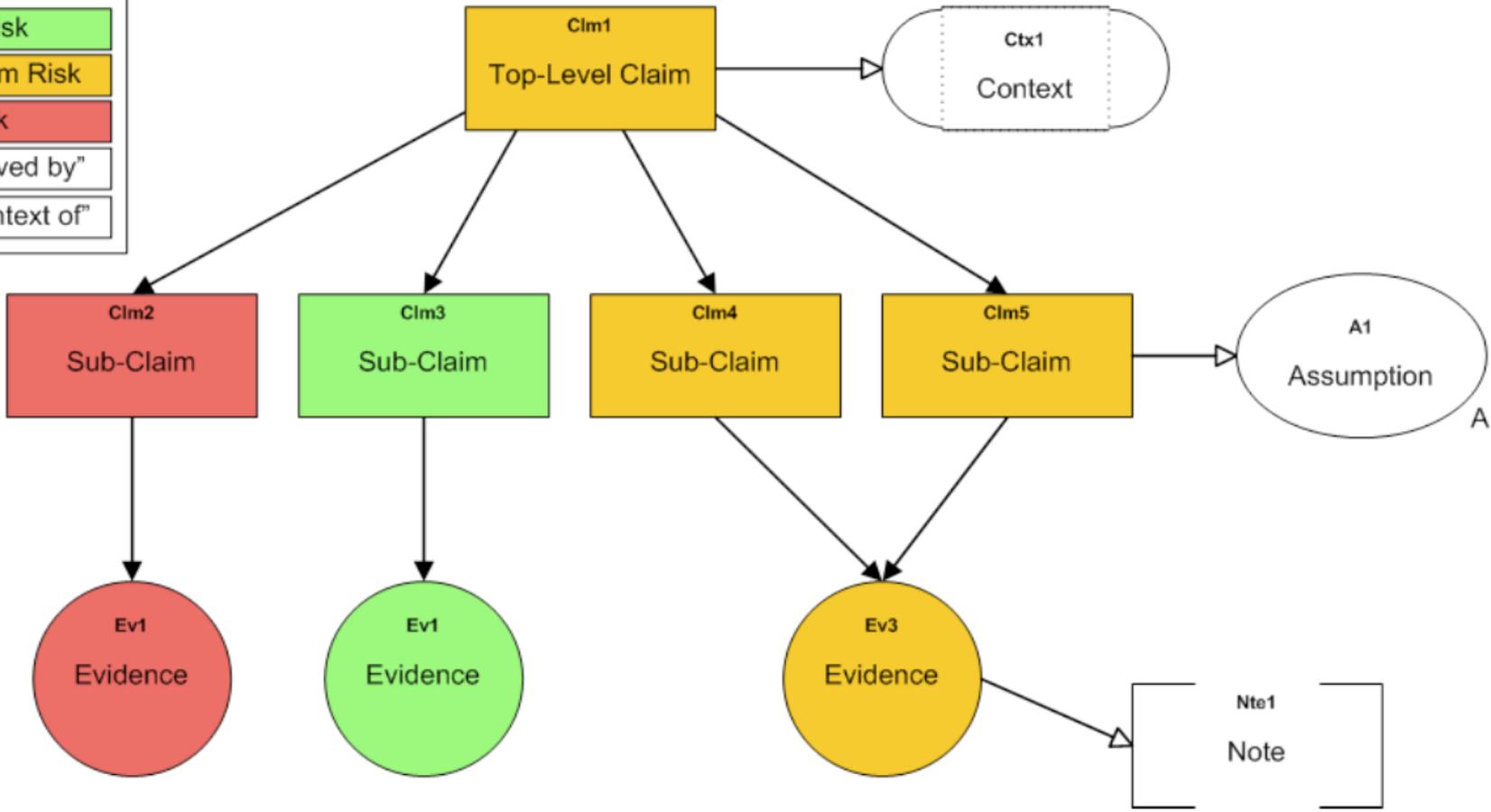
ISO/IEC 15026: Examples of relationships among parts



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

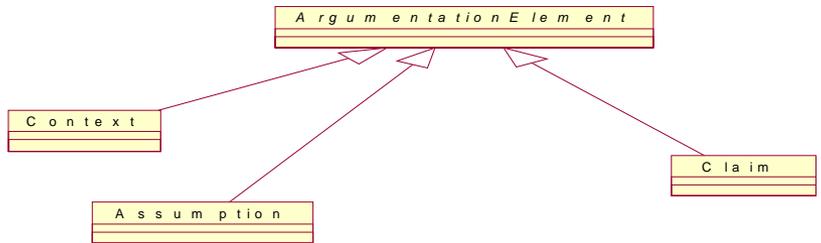
Legend:

- Green = Low Risk
- Yellow = Medium Risk
- Red = High Risk
- = "Is solved by"
- ▷ = "In context of"

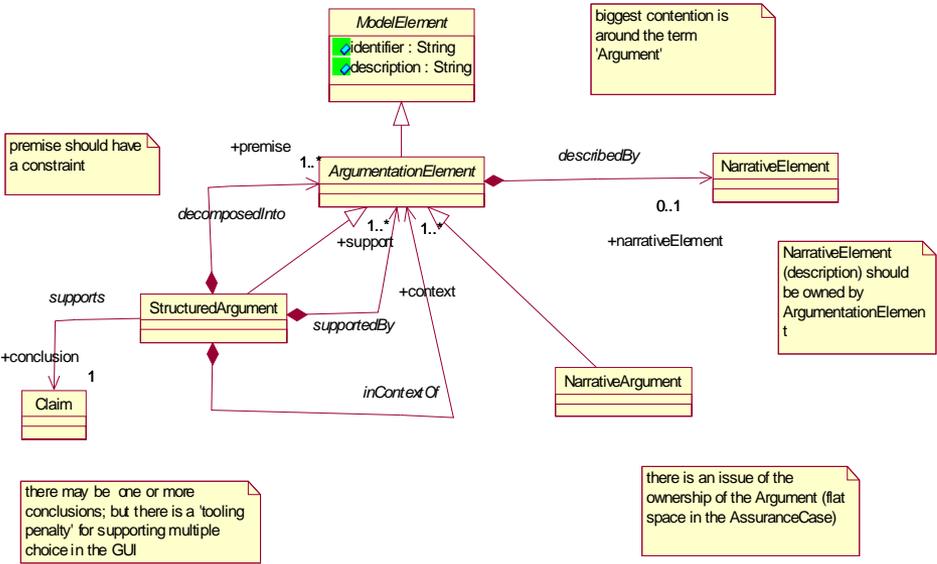


The Assurance Case/Argument: OMG Evidence and Claims/Arguments Standards

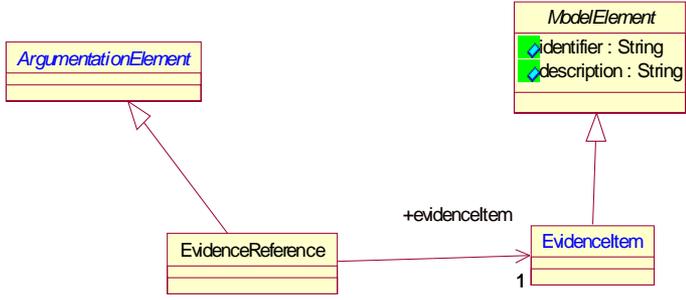
ARM:Claims



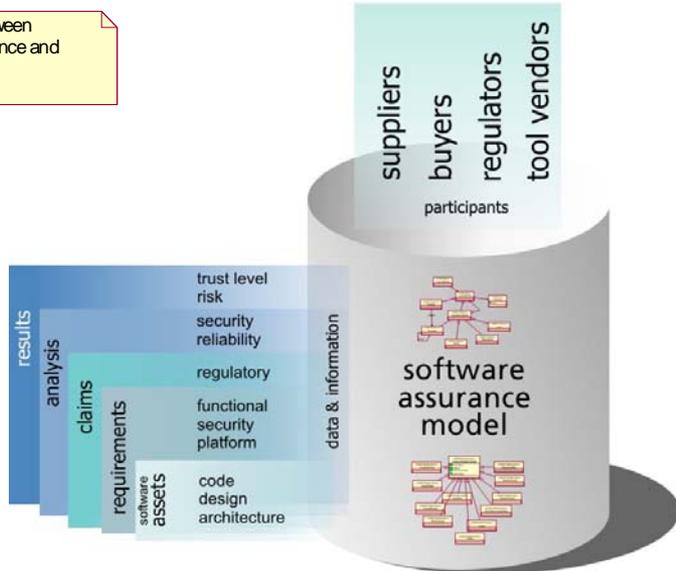
ARM:Arguments



SAEM: Evidence



duplication between EvidenceReference and EvidenceItem



DHS's Build Security In and SwA Websites

US-CERT: Software Assurance

Friday, February 08, 2008

Home | FAQ | Contact | Privacy & Use

US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

Security Publications | Alerts and Tips | Related Resources | About Us

Information For

- Technical
- Non-Technical
- Government
- Control Systems

Sign Up

Mailing Lists & Feeds

Reporting

Report an Incident

Report Phishing

Report a Vulnerability

DHS Threat Advisory

National Threat Advisory:
ELEVATED
Significant Risk of Serious Results

The threat level in the airline sector is High or Orange. Read more

Software Assurance

DHS Cyber Security

Software is essential to the operation of the Nation's critical intellectual property, consumer trust, and business operations and infrastructure, from process control systems to reliable software.

It is estimated that 90 percent of reported security incidents are due to software vulnerabilities, and reducing overall risk to cyber attacks is critical to include provisions for built-in security of the end user.

Setting a Higher Standard for Software Assurance

Grounded in the National Strategy to Secure Cyberspace, the Software Assurance Program spearheads the development of practical guidance in cyber security. Significant new research on secure software development issues from new methods that avoid basic problems when portions of the system software are compromised.

Through these efforts, Homeland Security seeks to reduce the risk of software vulnerabilities by improving the routine development and deployment of more secure and reliable software that supports mission-critical infrastructure.

From Patch Management to Software Assurance

The key objective of the Software Assurance Program is to shift the focus of software assurance. This shift is designed to encourage software developers to build security into their software from the start, rather than relying on applying patches to existing software.

Recognizing that software security is fundamentally a software development issue, the Software Assurance Program, in partnership with the public sector and private industry, to raise the standard of software security and academia will raise expectations for product assurance and security methodologies and tools as a normal part of business operations.

Building Success through Collaboration

www.us-cert.gov/swa/

Software Assurance

Community Resources and Information Clearinghouse

Sponsored by DHS National Cyber Security Division

HOME | PEOPLE | PROCESS | TECHNOLOGY | ACQUISITION | WORKING GROUPS

Software Assurance

What is Software Assurance?

Events

Resources

Working Groups

Workforce Education & Training

Processes & Practices

Technology, Tools & Product Eval.

Acquisition & Outsourcing

Measurement

Business Case

Malware

Join a Working Group

US-CERT Software Assurance

Build Security In

Resources to help you build security into your systems in every phase of development.

What is Software Assurance?

Software assurance (SwA) is the process of identifying, preventing, and mitigating software vulnerabilities, either internally or externally, at any time during its life cycle (from CNSS 4009 IA Glossary).

As part of the DHS risk mitigation strategy, the Software Assurance Program focuses on the development of trustworthy software through the development of trustworthiness diagnostic capabilities to address the following areas:

FOCUS AREAS

The SwA framework encourages the production, evaluation, and acquisition of better quality and more secure software, providing focuses in these four areas:

- People:** Education and training for developers and users
- Process:** Sound practices, standards, and practical guidelines for the development of secure software
- Technology:** Diagnostic tools, cyber security R&D and measurement
- Acquisition:** Specifications and guidelines for acquisition and outsourcing

The Software Assurance Forum and several working groups composed of volunteers from government, industry, and academia are helping the Software Assurance Program achieve its objectives in these focus areas.

Software Assurance Focus Area and Working Group Matrix

Working Group	People	Process	Technology	Acquisition
Workforce Education and Training	●	●	●	●
Processes and Practices	●	●	●	●
Technology, Tools, and Product Evaluation	●	●	●	●
Acquisition and Outsourcing	●	●	●	●
Measurement	●	●	●	●
Business Case	●	●	●	●
Malware	●	●	●	●

WHY IS SOFTWARE ASSURANCE CRITICAL?



[makingsecuritymeasurable.mitre.org]



A Collection of Information Security Community Standardization Activities and Initiatives

Home | Current Collection | Feedback Requested

Measurable security pertains at a minimum to the following areas:

- Vulnerability Management
- Asset Security Assessment
- Configuration Guidance
- Malware Response
- Threat Analysis
- Intrusion Detection
- Asset Management
- Patch Management
- Incident Management

MITRE, in collaboration with government, industry, and academic stakeholders, is improving the measurability of security through enumerating baseline security data, providing standardized languages as means for accurately communicating the information, and encouraging the sharing of the information with users by developing repositories.

The other activities and initiatives listed here have similar concepts or compatible approaches to MITRE's. Together all of these efforts are helping to make security more measurable by defining the concepts that need to be measured, providing for high fidelity communications about the measurements, and providing for sharing of the measurements and the definitions of what to measure.

Enumerations | Languages | Repositories

- CVE** - Common Vulnerabilities and Exposures (CVE®) - common vulnerability identifiers
- CWE** - Common Weakness Enumeration (CWE™) - list of software weakness types
- CAPEC** - Common Attack Pattern Enumeration and Classification (CAPEC™) - list of common attack patterns
- CME** - Common Malware Enumeration (CME™) - common identifiers for viruses, worms, and other malicious code
- CCE** - Common Configuration Enumeration (CCE™) - common security configuration identifiers
- CPE** - Common Platform Enumeration (CPE™) - common platform identifiers
- SANS Top Twenty** - SANS/FBI consensus list of the Twenty Most Critical Internet Security Vulnerabilities that uses CVE-IDs to identify the issues
- OWASP Top Ten** - ten most critical Web application security flaws
- WASC Web Security Threat Classification** - list of Web security threats

- OVAL** - Open Vulnerability and Assessment Language (OVAL®) - standard for determining vulnerability and configuration issues
- CRF** - Common Result Format (CRF™) - standardized assessment result format for conveying findings based on common names and naming schemes
- CEE** - Common Event Expression (CEE™) - standardizes the way computer events are described, logged, and exchanged
- OVAL Interpreter** - free tool for collecting information for testing, carrying out OVAL Definitions, and presenting results of the tests
- Benchmark Editor™** - free tool that enhances and simplifies creation and editing of benchmark documents written in XCCDF and OVAL
- Extensible Configuration Checklist Description Format (XCCDF)** - specification language for uniform expression of security checklists, benchmarks, and other configuration guidance
- Common Vulnerability Scoring System (CVSS)** - open standard that conveys vulnerability severity and helps determine urgency and priority of response
- Common Announcement Interchange Format (CAIF)** - XML-based format created to store and exchange security announcements in a normalized way
- OMG Semantics of Business Vocabulary and Business Rules (SBVR)** - language for interchange of business vocabularies and rules among organizations and software tools

- OVAL Repository** - community-developed OVAL Vulnerability, Compliance, Inventory, and Patch Definitions
- National Vulnerability Database (NVD)** - U.S. vulnerability database based on CVE that integrates all publicly available vulnerability resources and references
- NIST Security Content Automation Protocol (SCAP)** - security content for automating technical control compliance activities, vulnerability checking, and security measurement
- Red Hat Repository** - OVAL Patch Definitions corresponding to Red Hat Errata security advisories
- Center for Internet Security (CIS) Benchmarks** - best-practice security configurations accepted for compliance with FISMA, the ISO standard, GLB, SOX, HIPAA, and FIRPA, and other regulatory requirements for information security
- DISA Security Technical Implementation Guides (STIGS)** - U.S. Defense Information Systems Agency's (DISA) STIGS are configuration standards for DOD information assurance and information assurance-enabled devices and systems

View the [current collection](#) of organizations, activities, and initiatives.

Disclaimer

This Web site is hosted by The MITRE Corporation, © 2008 The MITRE Corporation. CVE is a registered trademark and the Making Security Measurable logo, CCE, CME, CWE, CPE, and OVAL are trademarks of The MITRE Corporation. All other trademarks are the property of their respective owners. Contact us: measurablesecurity@mitre.org

Page Last Updated: January 17, 2008

The image features a solid blue background. In the foreground, there are dark silhouettes of a dog and a cat. The dog is positioned in the upper half, facing right with its tail curved upwards. The cat is in the lower half, also facing right. The word "Questions?" is written in a bold, white, sans-serif font, centered horizontally and partially overlapping the dog's body.

Questions?