



Homeland
Security



Commerce



National
Defense



Training & Software Security Engineering: CWE

Knowing what could make
software vulnerable to attack

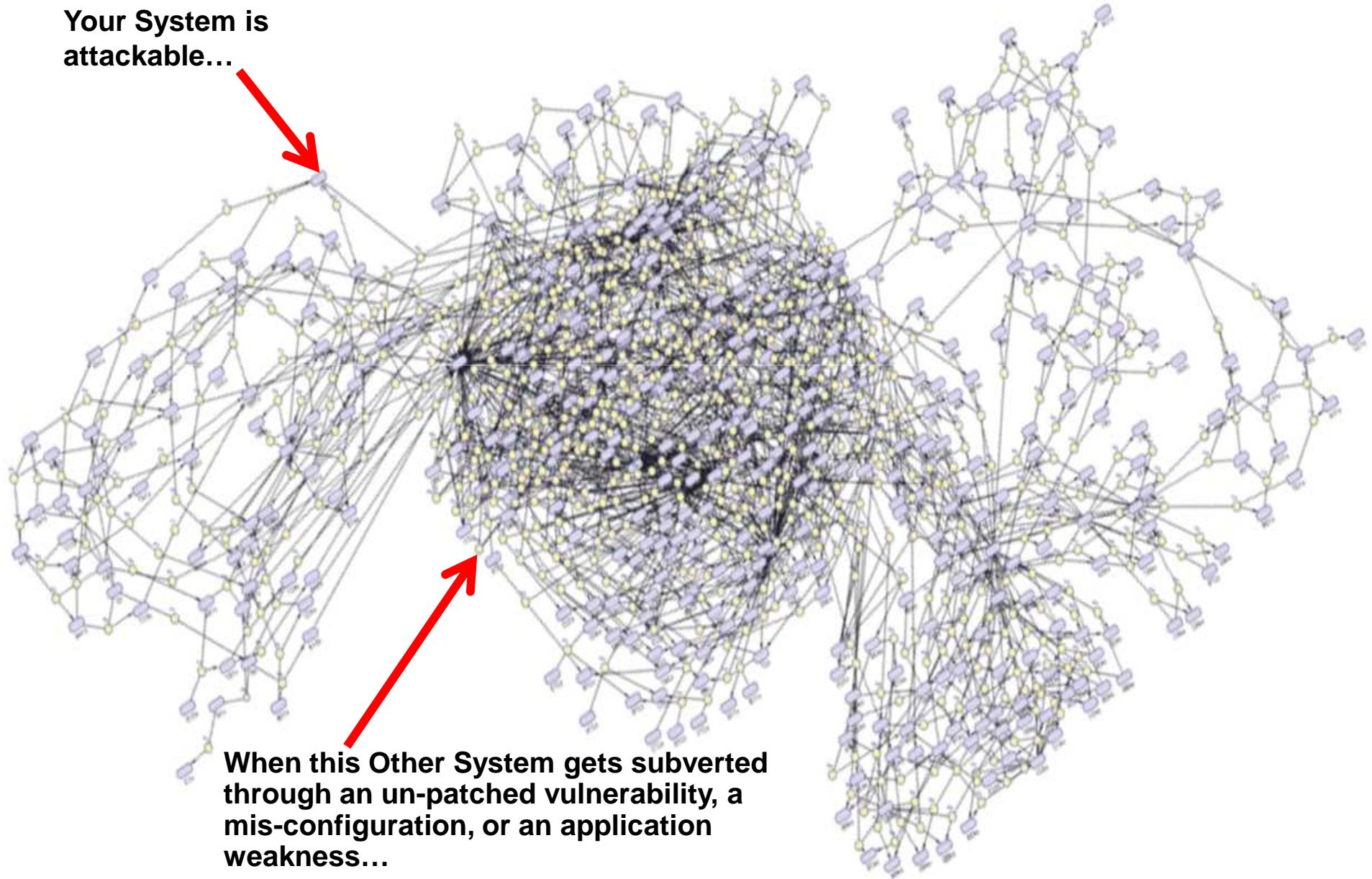


Robert A. Martin
28 February 2011



Today Everything's Connected

Your System is
attackable...

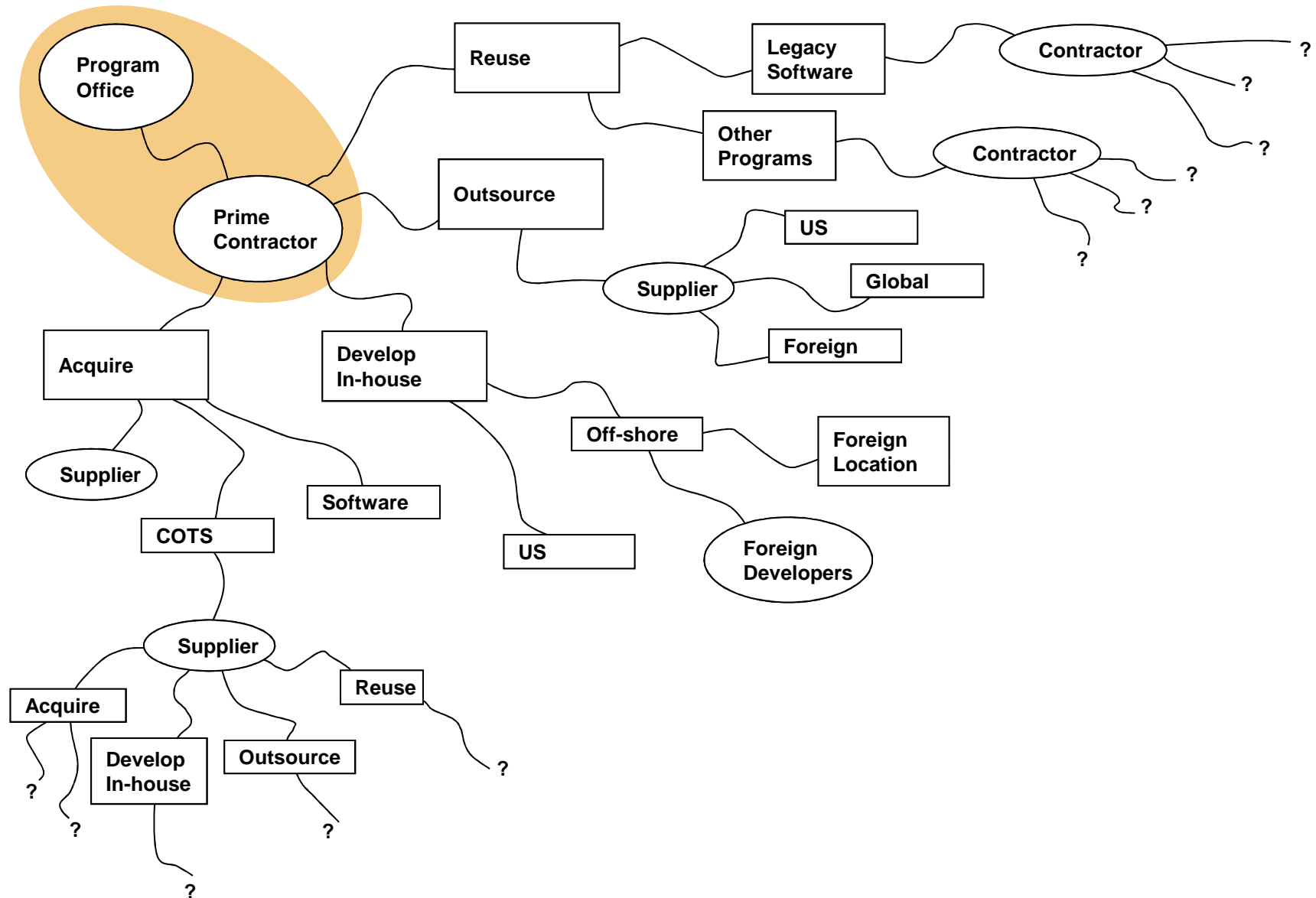


When this Other System gets subverted
through an un-patched vulnerability, a
mis-configuration, or an application
weakness...



The Software Supply Chain

*

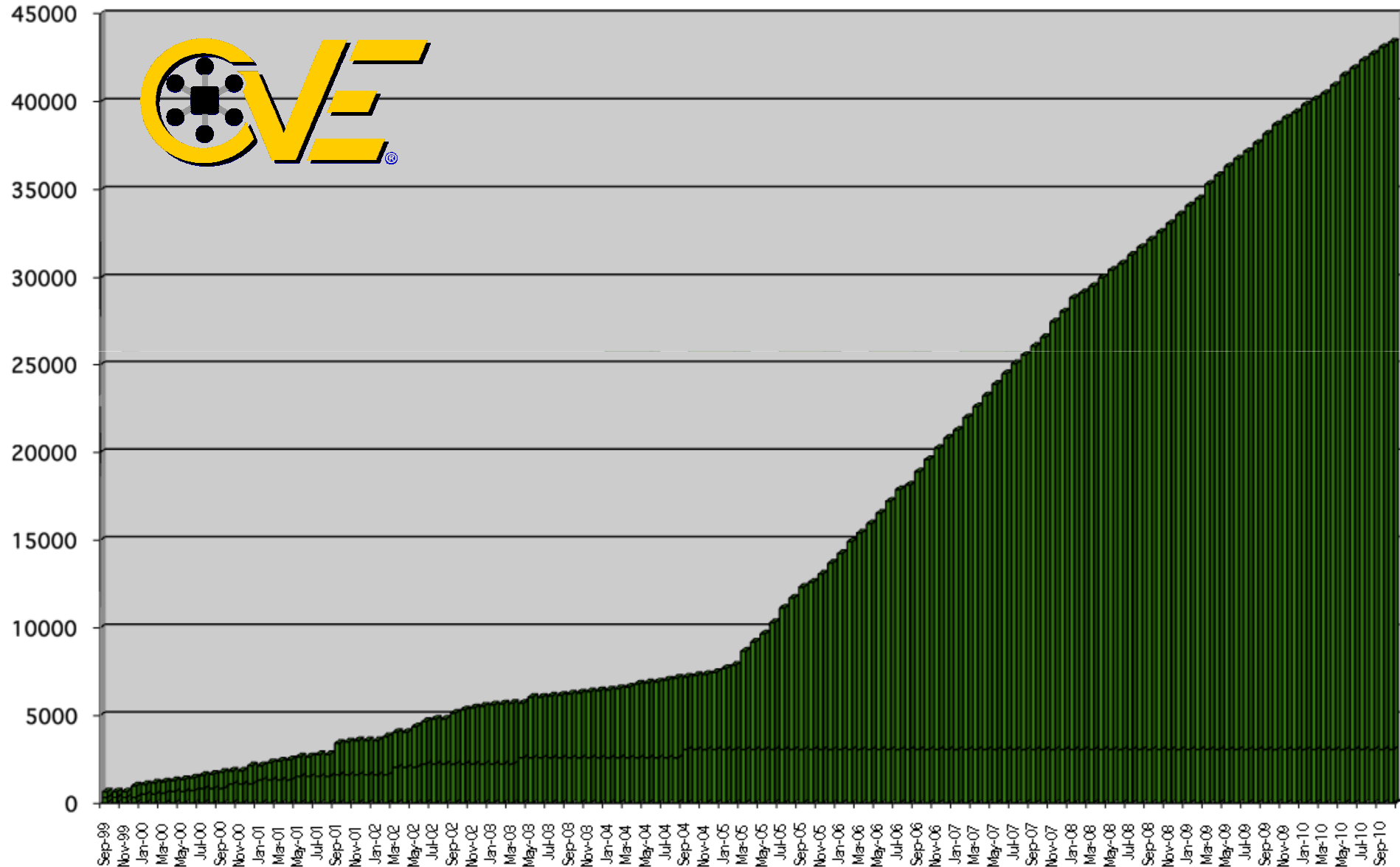


* “Scope of Supplier Expansion and Foreign Involvement” graphic in DACS www.softwaretchnews.com Secure Software Engineering, July 2005 article “Software Development Security: A Risk Management Perspective” synopsis of May 2004 GAO-04-678 report “Defense Acquisition: Knowledge of Software Suppliers Needed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

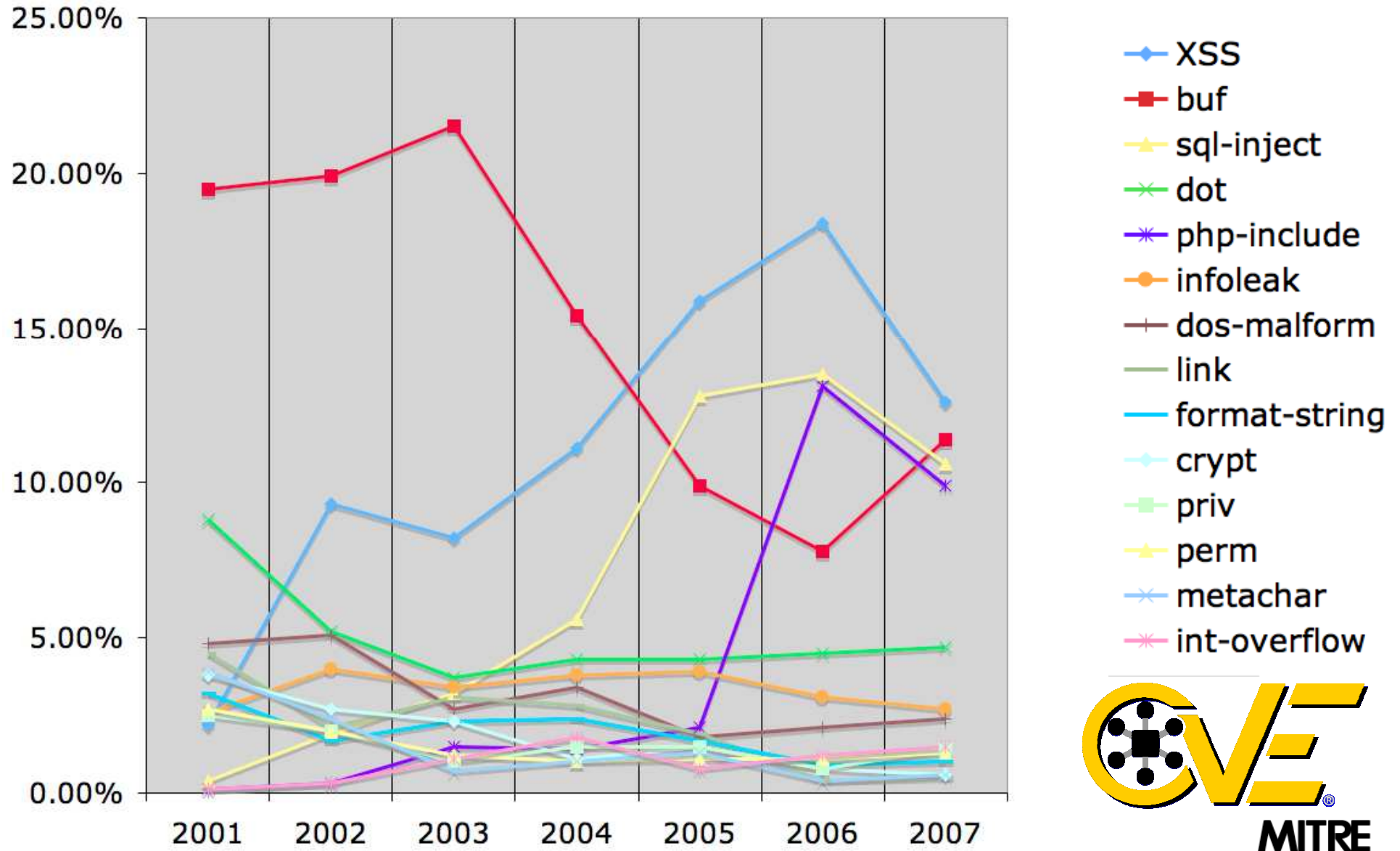
**If the weaknesses
in software were as
easy to spot and
their impact as
obvious as...**



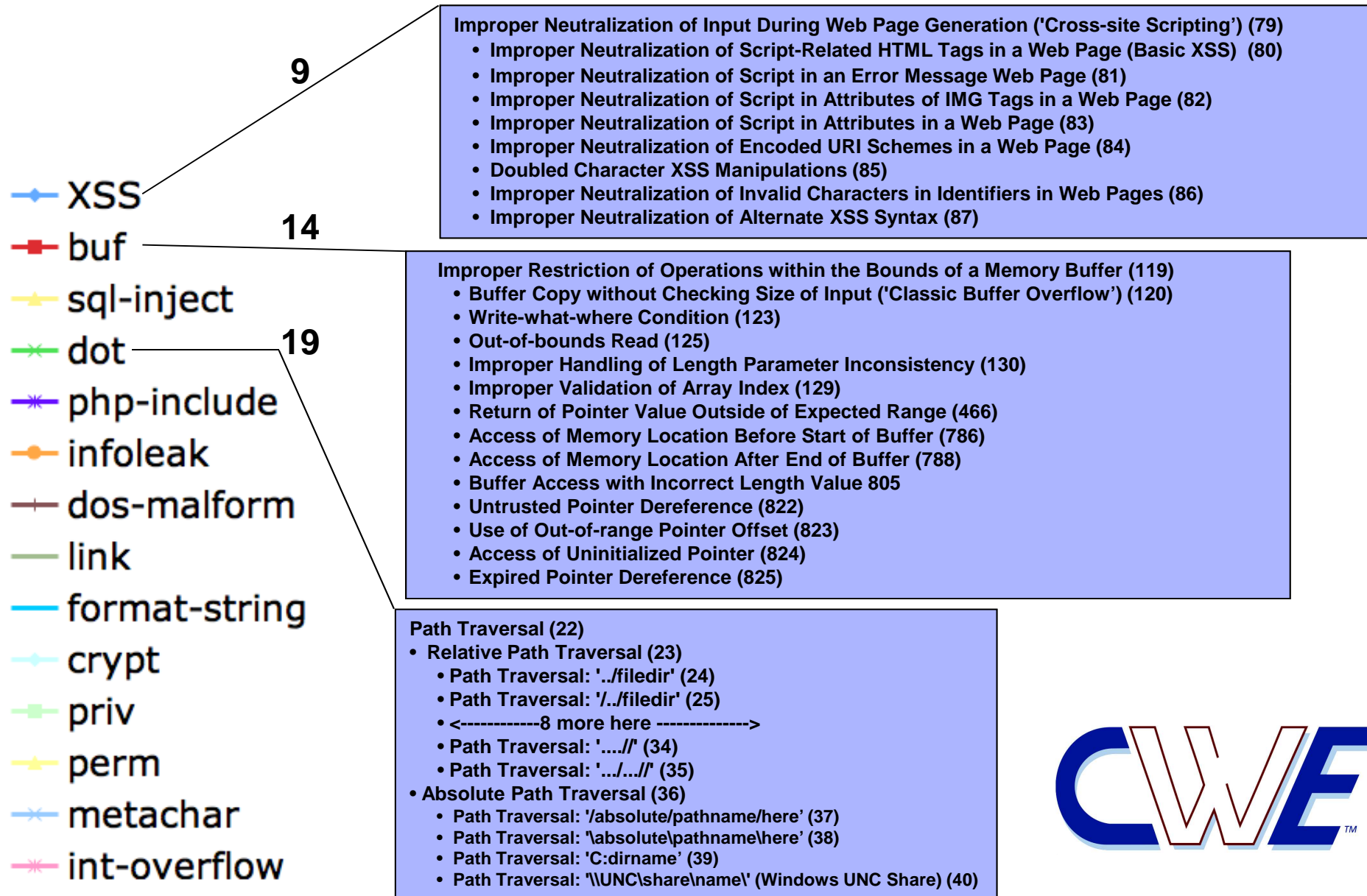
CVE 1999 to 2011



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



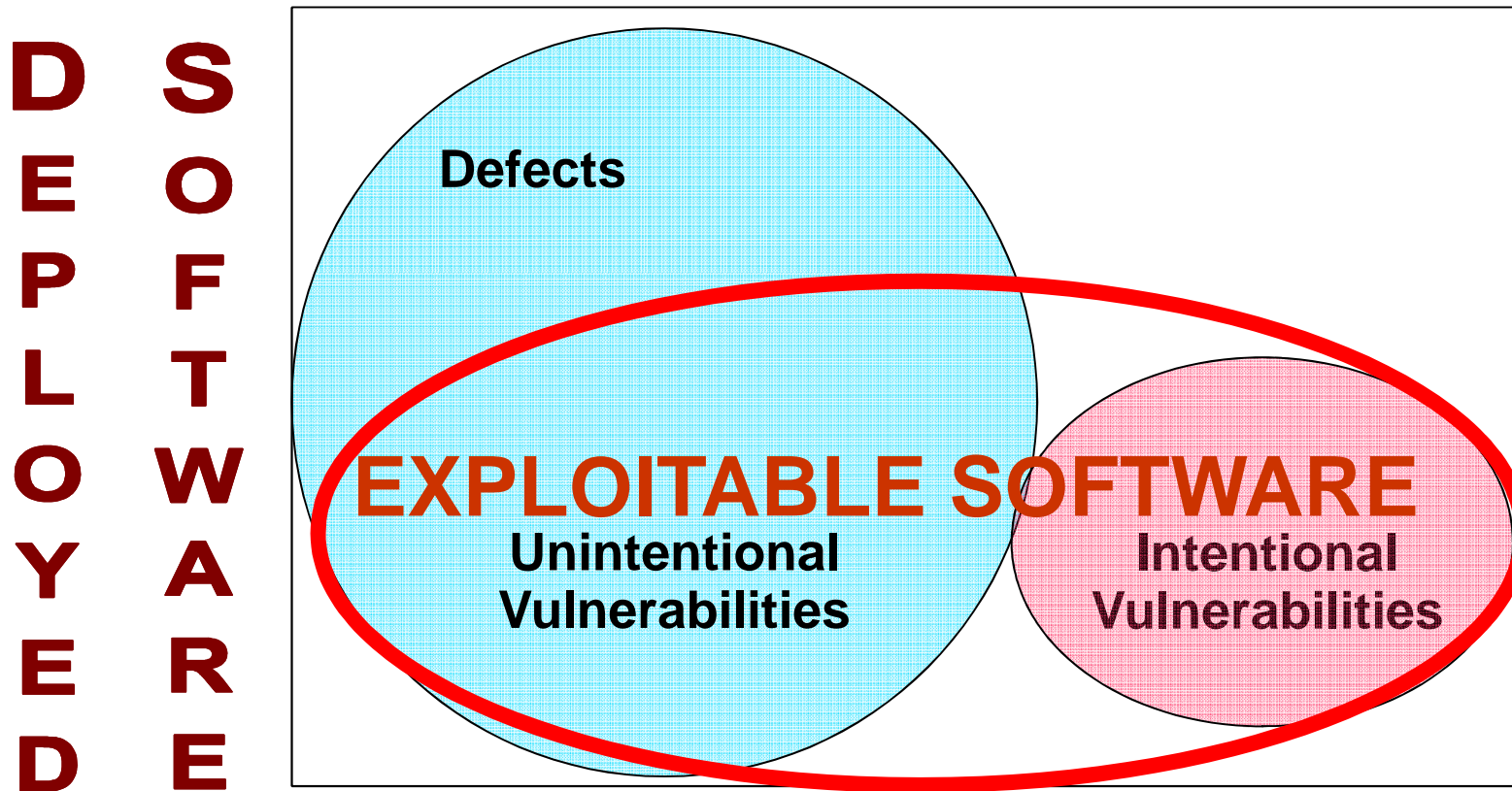
Removing and Preventing the Vulnerabilities Requires More Specific Definitions...CWEs



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.



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

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

Common Weakness Enumeration (CWE)

- dictionary of weaknesses
 - weaknesses that can lead to exploitable vulnerabilities (i.e. CVEs)
 - the things we don't want in our code, design, or architecture
 - web site with XML of content, sources of content, and process used
- structured views
 - provides multiple views into CWE dictionary content
 - supports alternate views – developer/researcher/sub-views
- open community process
 - to facilitate common terms/ concepts/facts and understanding
 - allows for vendors, developers, system owners and acquirers to understand tool capabilities/ coverage and priorities
 - utilize community expertise

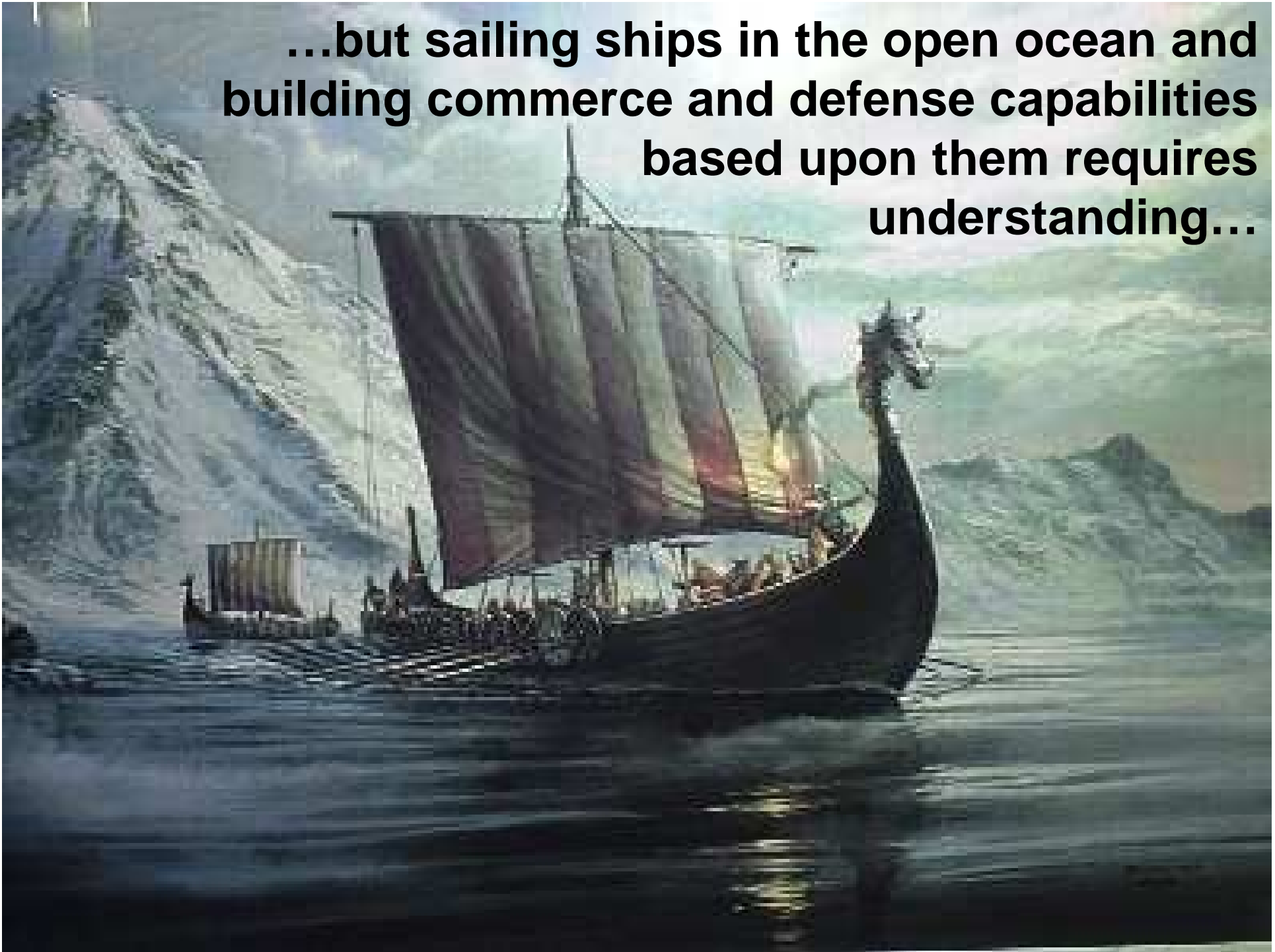
**Foundation for
other DHS, NSA,
OSD, NIST, OWASP,
SANS, and OMG
SwA Efforts**



Building **Software**
only require a few
skills and basic
understanding...



...but sailing ships in the open ocean and building commerce and defense capabilities based upon them requires understanding...



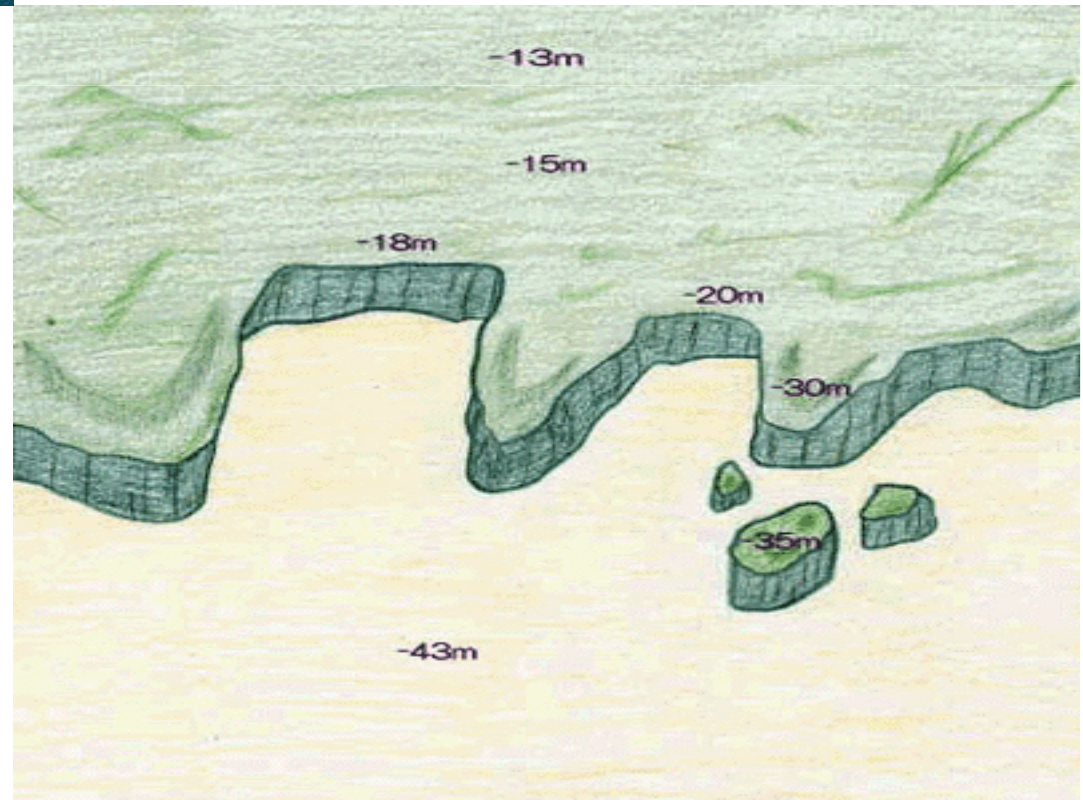
...of navigation threats...

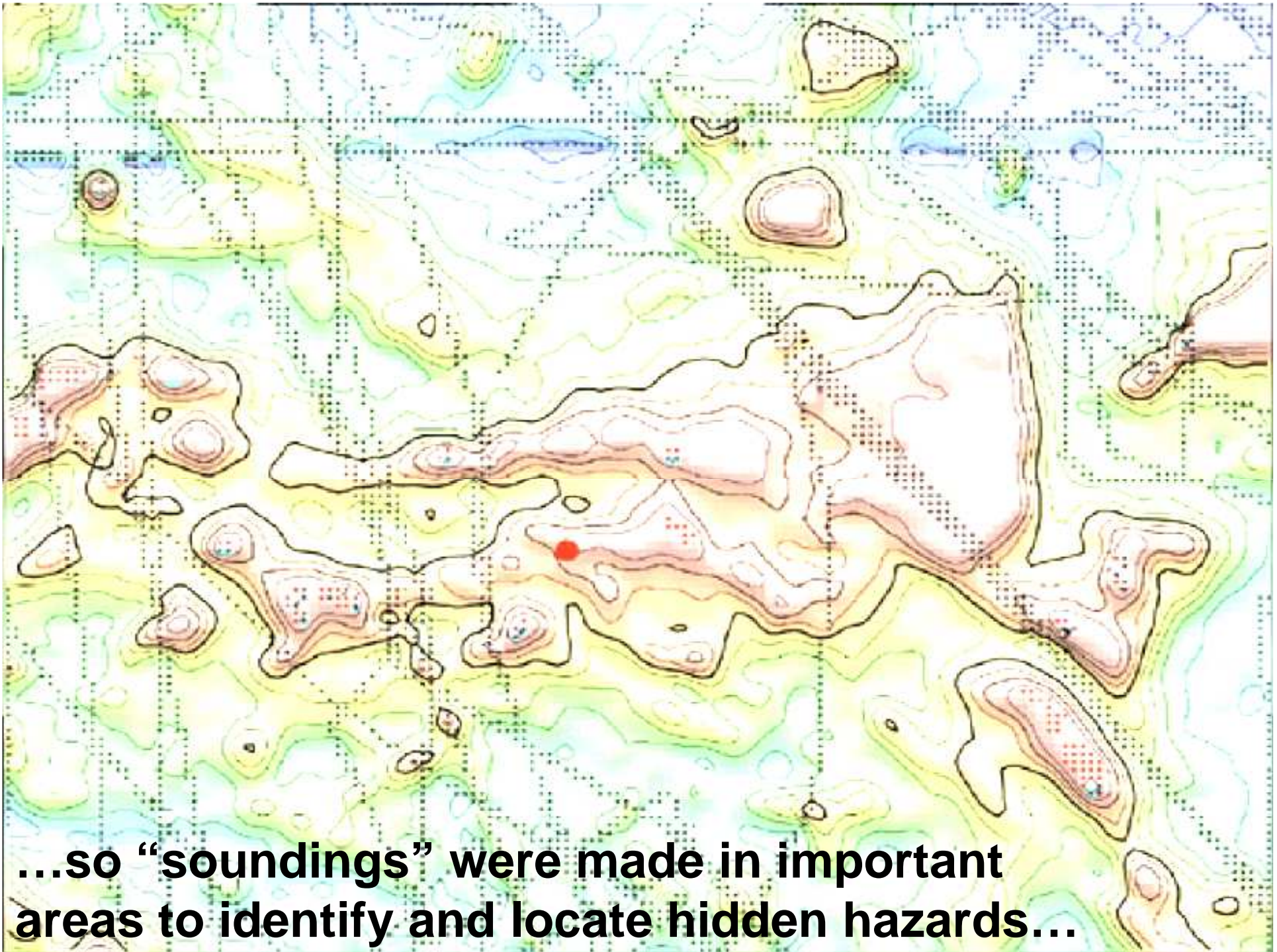




...surface maps didn't capture the full set of threats and hazards – i.e. what was really going on...

...a more insightful depiction – one that shows what was going on under the surface – was needed...





...so “soundings” were made in important areas to identify and locate hidden hazards...

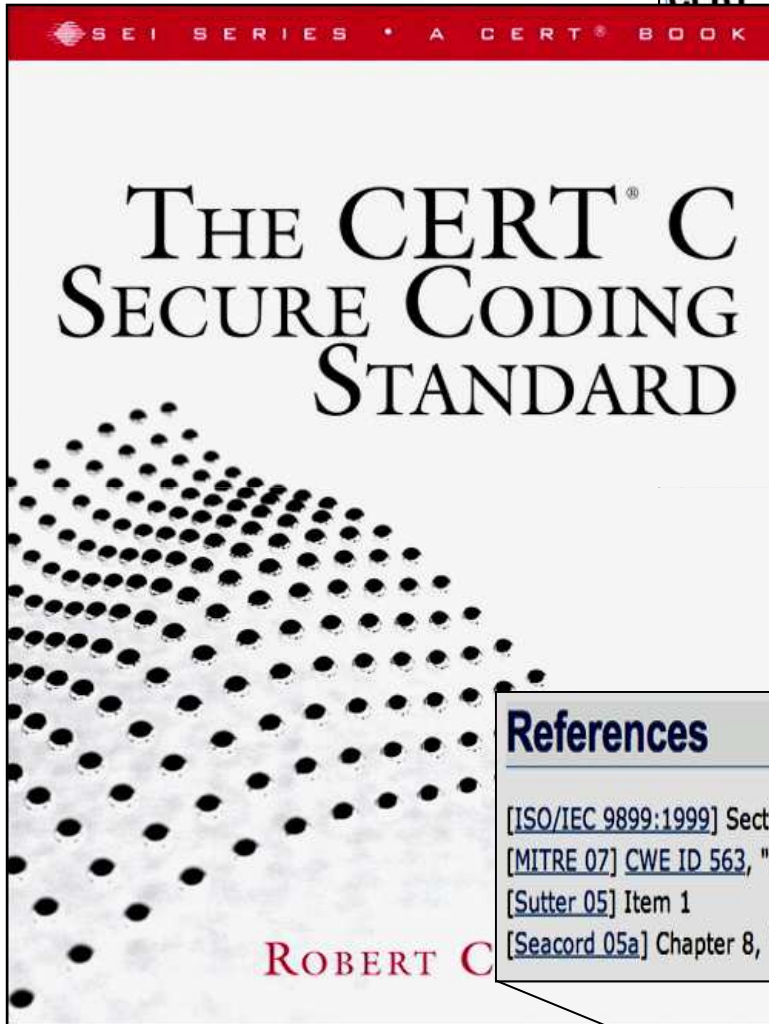
...and warning signals to help others avoid known hazards were erected along with...



...indicators showing safe ways to avoid the known hazards...







MSC00-CPP. Compile cleanly at high warning levels - CERT Secure Coding Standards

https://www.securecoding.cert.org/confluence/display/cplusplus/MSC00-CPP.+Compile+cleanly+at+high+warning+levels

CERT

Software Assurance Secure Systems Organizational Security Coordinated Response Training

C++ Secure Coding Practices

MSC00-CPP. Compile cleanly at high warning levels

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

Labels: [undefined](#) [implementation-defined](#)

Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.

According to C99 [ISO/IEC 9899:1999] Section 5.1.1.3:

A conforming implementation shall produce at least one diagnostic message (identified in an *implementation-defined* manner) if a preprocessing translation unit or translation unit contains a violation of any syntax rule or constraint, even if the behavior is also explicitly specified as *undefined* or *implementation-defined*. Diagnostic messages need not be produced in other circumstances.

Assuming a conforming implementation, eliminating diagnostic messages will eliminate any syntactic or constraint violations.

If suitable source code-checking tools are available, use them regularly.

Exceptions

MSC00-EX1: Compilers can produce diagnostic messages for correct code. This is permitted by C99 [ISO/IEC 9899:1999], which allows a compiler to produce a diagnostic for any reason. It is usually preferable to rewrite code to eliminate compiler warnings, but if the code is correct it is sufficient to provide a comment explaining why the warning message does not apply. Some compilers provide ways to suppress warnings, such as suitably formatted comments or pragmas, which can be used sparingly when the programmer understands the implications of the warning but has good reason to use the flagged construct anyway.

Do not simply quiet warnings by adding type casts or other means. Instead, understand the reason for the warning and consider a better approach, such as using matching types and avoiding type casts whenever possible.

Risk Assessment

Eliminating violations of syntax rules and other constraints can eliminate serious software vulnerabilities that can lead to the execution of arbitrary code with the permissions of the vulnerable process.

References

- [ISO/IEC 9899:1999] Section 5.1.1.3, "Diagnostics"
- [MITRE 07] [CWE ID 563](#), "Unused Variable"; [CWE ID 570](#), "Expression is Always False"; [CWE ID 571](#), "Expression is Always True"
- [Sutter 05] Item 1
- [Seacord 05a] Chapter 8, "Recommended Practices"

Related Sites

US-CERT

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

...but new types of threats and hazards can occur in unexpected places and in new ways and...

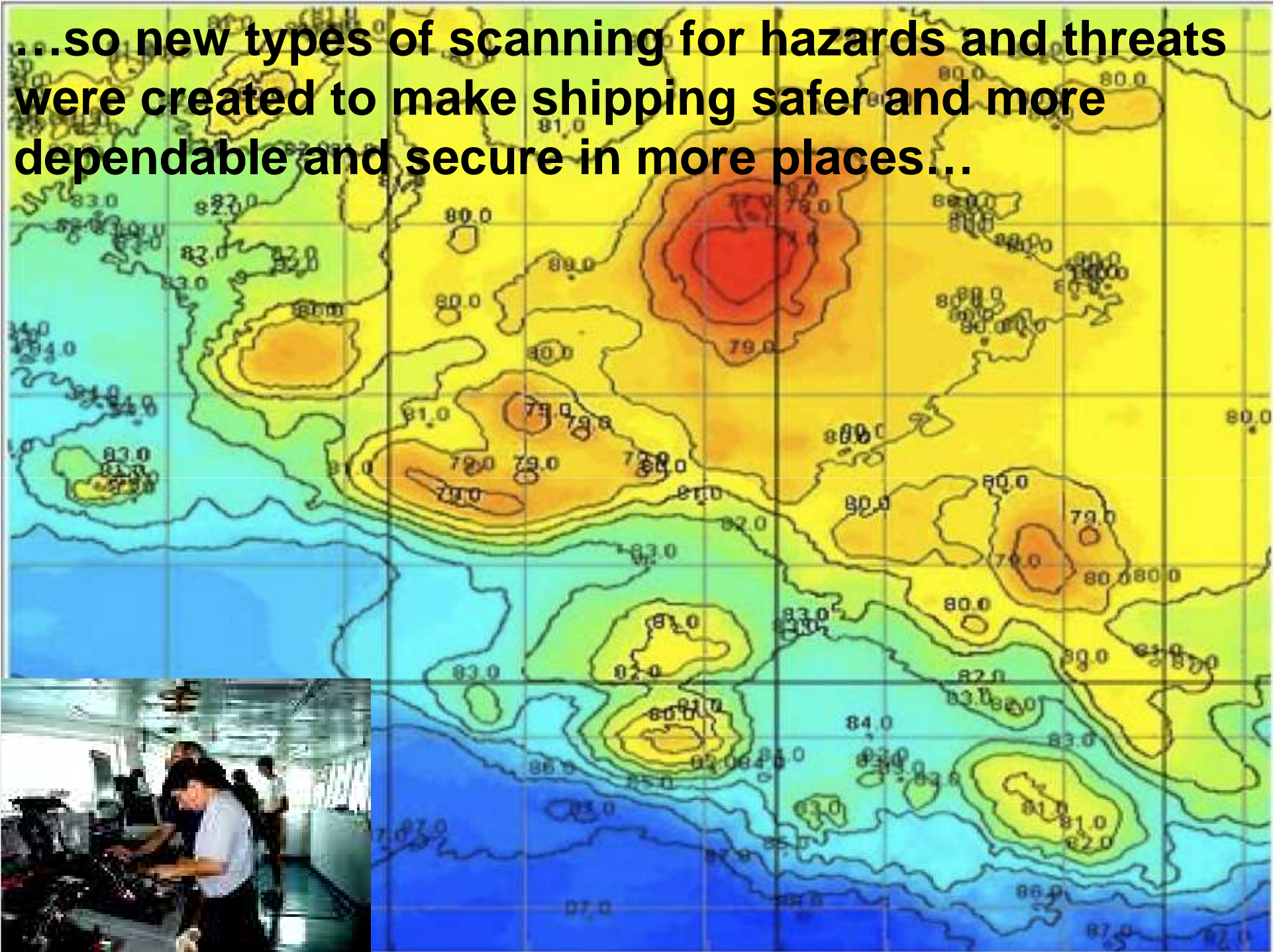




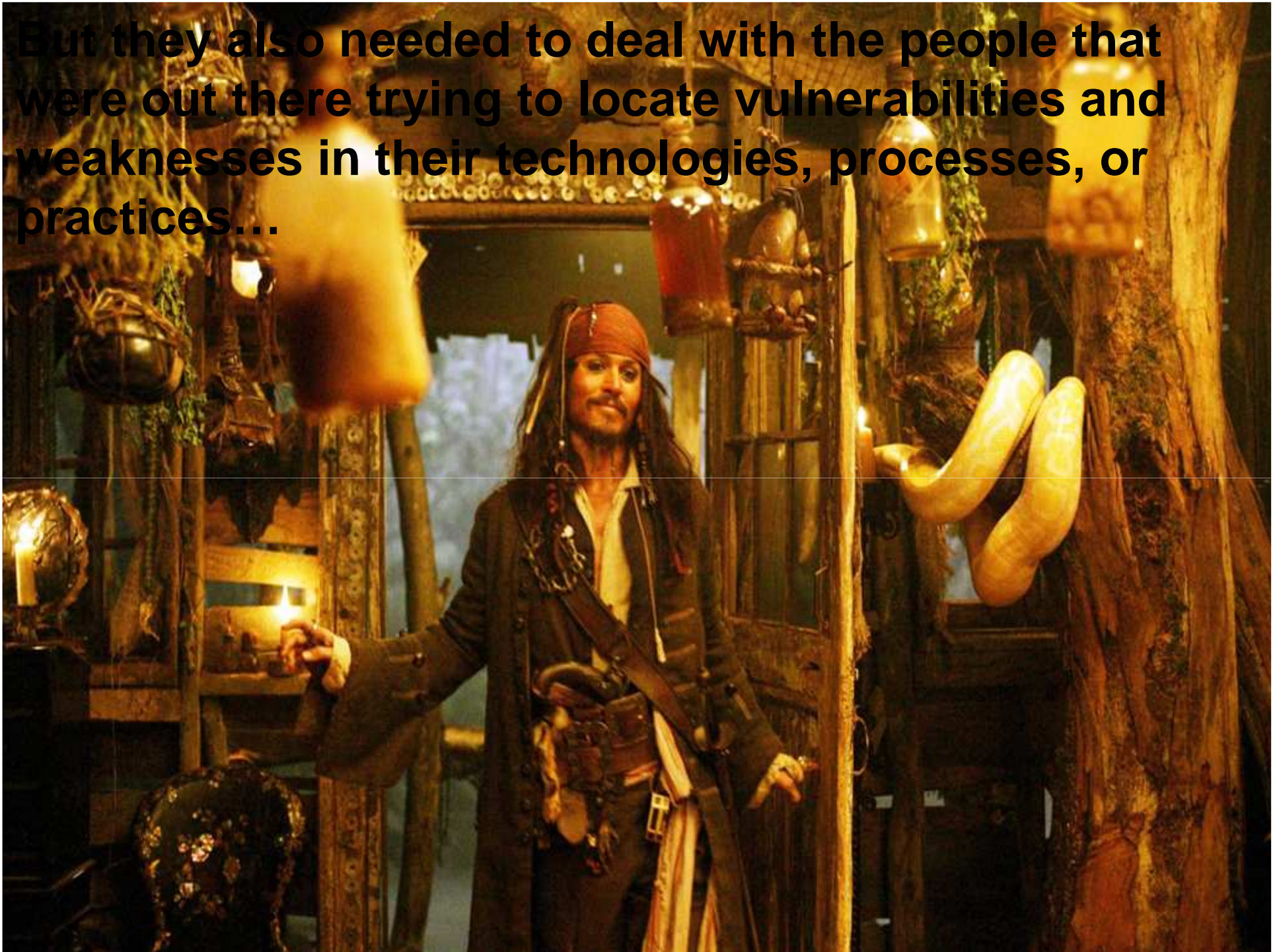
...some threats and hazards are unpredictable and dynamic...



...so new types of scanning for hazards and threats were created to make shipping safer and more dependable and secure in more places...



But they also needed to deal with the people that were out there trying to locate vulnerabilities and weaknesses in their technologies, processes, or practices...



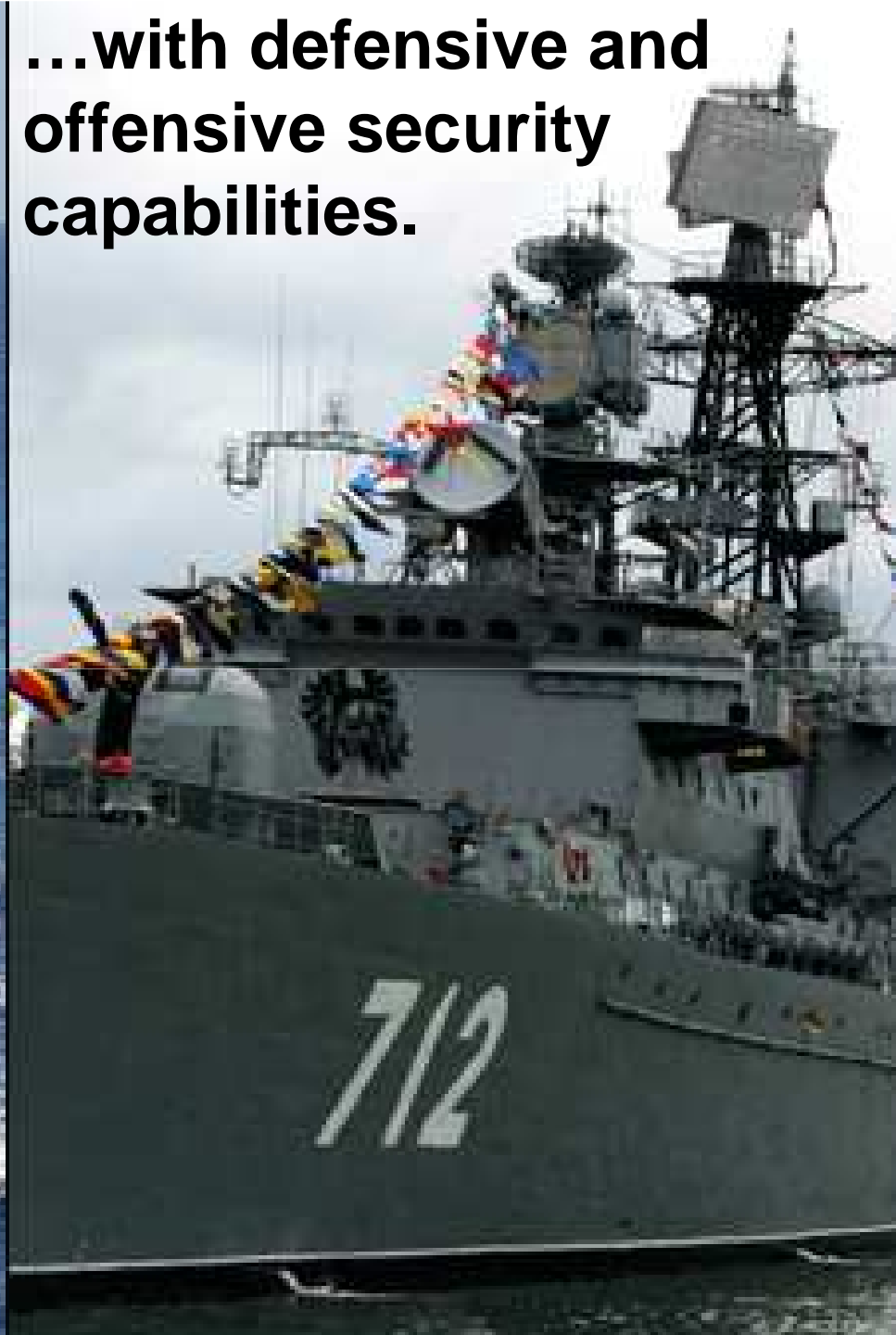


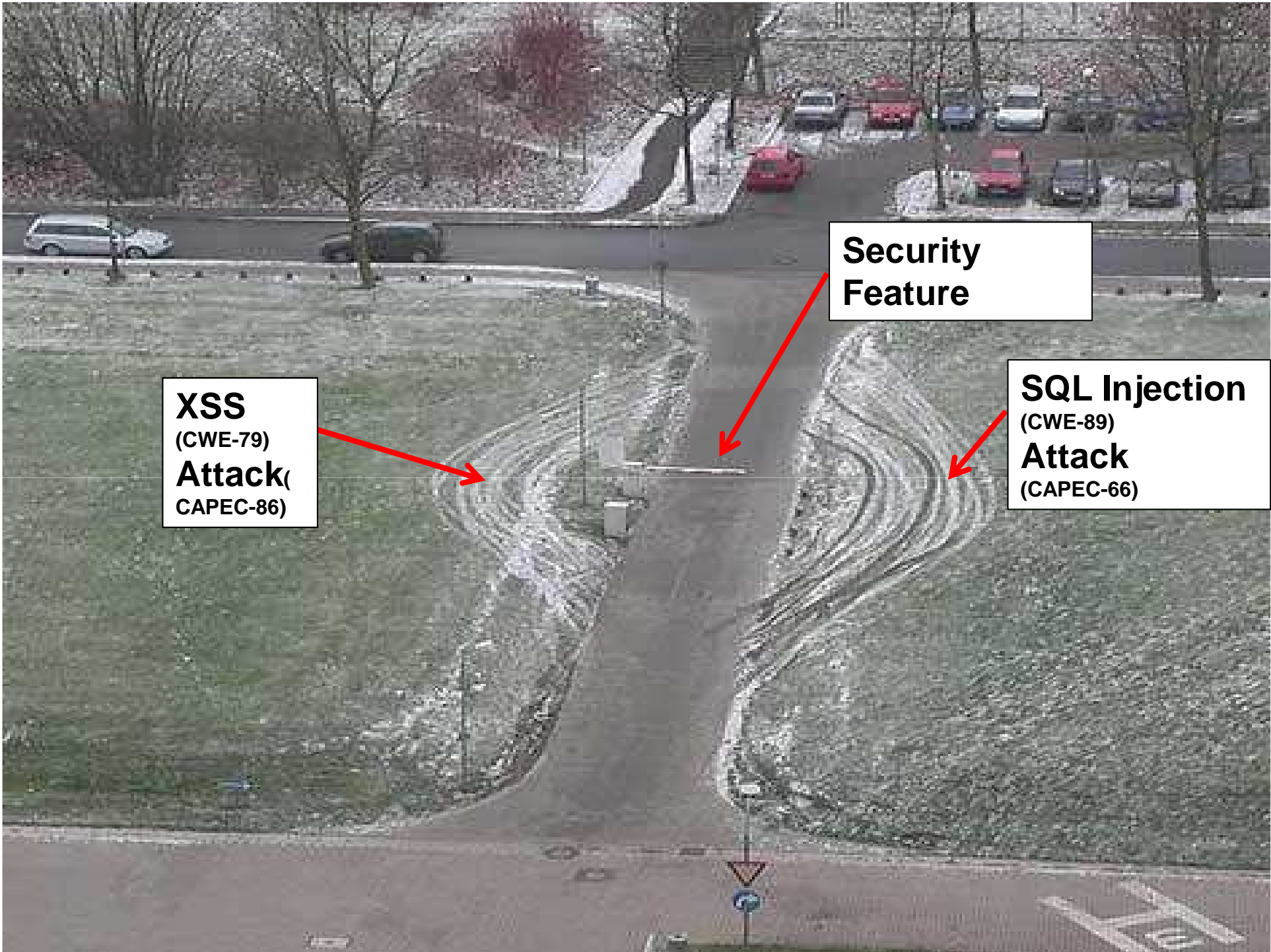
**...and so they
also had to deal
with active and
intelligent
threats...**





...with defensive and offensive security capabilities.





XSS
(CWE-79)
Attack
(CAPEC-86)

Security
Feature

SQL Injection
(CWE-89)
Attack
(CAPEC-66)

Software [In]security: Cyber Warmongering and Influence Peddling



By [Gary McGraw](#) and [Ivan Arce](#)


Nov 24, 2010

Article is provided courtesy of Addison-Wesley Professional

“For years in computer security, we have been attempting to protect the broken stuff from the bad people by placing a barrier between the bad people and the broken stuff. We have failed. Instead, we need to fix the broken stuff so that attacking it successfully takes far more resources and skill than is currently the case.”

CWE - Common Weakness Enumeration

http://cwe.mitre.org/



CWE and SANS Institute

TOP 25

MOST DANGEROUS SOFTWARE ERRORS

Search by ID:

CWE List

Full Dictionary View

Development View

Research View

Reports

About

Sources

Process

Documents

Community

Related Activities

Discussion List

Research

CWE/SANS Top 25

CWSS

News

Calendar

Free Newsletter

Compatibility

Program

Requirements

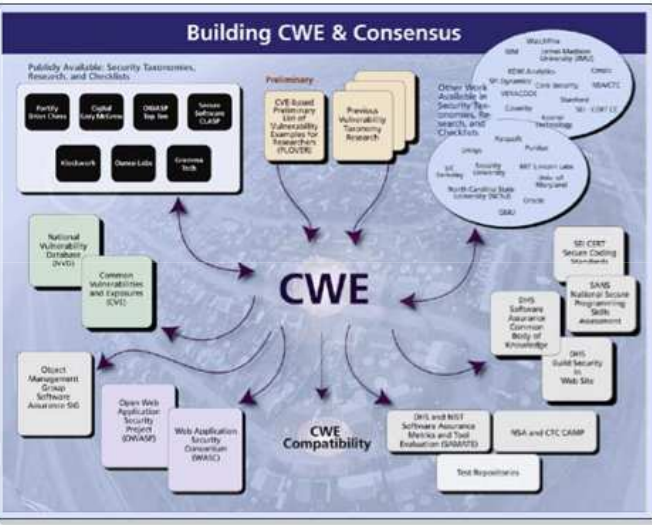
Declarations

Make a Declaration

Contact Us

Search the Site

International in scope and free for public use, CWE™ provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design.



Similar Standards

<p>Attack Patterns (CAPEC)</p> <p>Vulnerabilities (CVE)</p> <p>Configurations (CCE)</p> <p>Platforms (CPE)</p> <p>Malware (MAEC)</p>	<p>Assessment Language (OVAL)</p> <p>Checklist Language (XCCDF)</p> <p>Log Format (CEE)</p> <p>Security Content Automation (SCAP)</p> <p>Making Security Measurable</p>
--	---

News

- [Updated Common Weakness Scoring System \(CWSS\) White Paper Now Available](#)
- [LDRA Makes Two Declarations of CWE Compatibility](#)
- [Software Assurance keynote and Making Security Measurable table booth at International Conference on Software Quality](#)
- [CWE/Making Security Measurable booth at Black Hat DC 2011](#)

...more

Upcoming Events

- [CWE/Making Security Measurable booth at RSA 2011, February 14-18](#)
- [CWE/CAPEC/MAEC briefings at DHS/DoD/NIST SwA Forum, February 28 - March 4](#)
- [CWE/Making Security Measurable booth at 2011 Information Assurance Symposium, March 8-10](#)

...more

Status Report

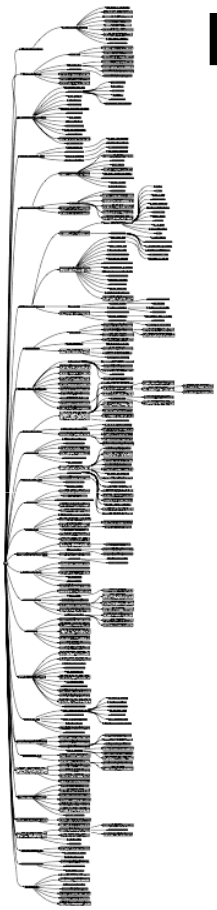
Version 1.11 posted December 13, 2010. 7 new entries were created, mostly related to synchronization and "functionality inclusion." One entry was deprecated. There are changes to 135 entries, especially potential mitigations, names, descriptions, demonstrative examples, and relationships. There were no schema changes.

More Information

cwe@mitre.org

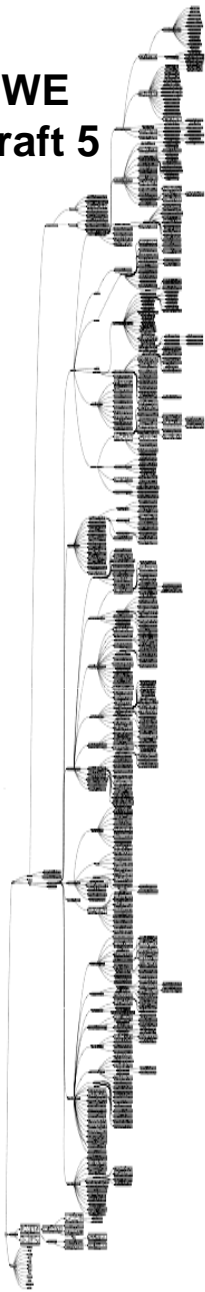


**PLOVER
(CWE
draft 1)**



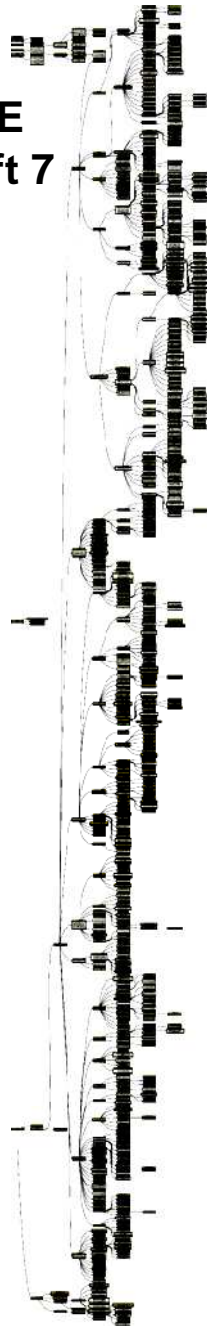
**2005
300 nodes**

**CWE
draft 5**



**2006
599 nodes**

**CWE
draft 7**



**2007
634 nodes**

**CWE
Vers
1.0**



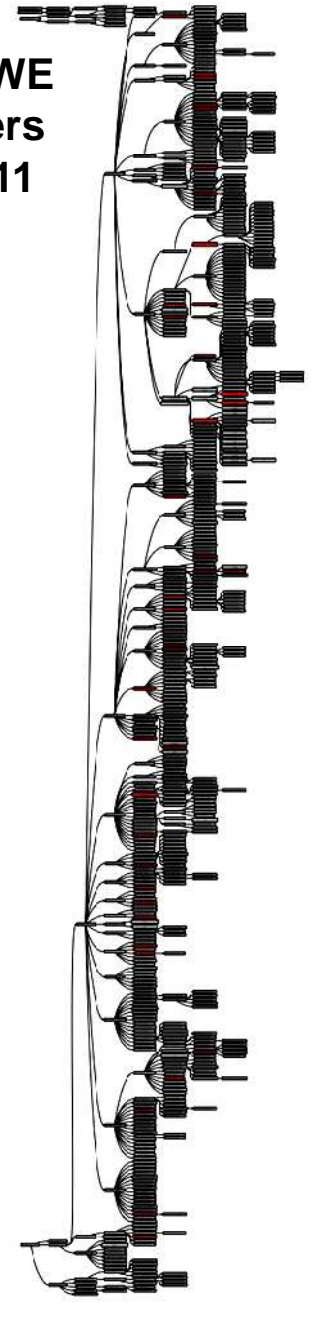
**2008
673 nodes**

**CWE
Vers
1.5**



**2009
799 nodes**

**CWE
Vers
1.11**



**Dec 2010
835 nodes**

CWE is Meant for People to Use



CWE Version 1.4

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

Project Lead:
Robert A. Martin



Security
Measurable™

CWE Version 1.4 Table of Contents

1
1
1
1
2
3
4
5
6
7
7
8
9
10
12
13
13
13
14
14
21
22
24
26
27
27
28
29
31
32
33
34
35
36
37
38
39
40
41
42
43
45
45
46
46
47
47
48
49
49
50
50
51
51
52
53
53
iii

CWE Version 1.4 CWE-1: Location

Status: Incomplete

roduced during the

Page
699
13
13
699
695

Status: Draft

onmental

Page
699
1
700
2
700
3
700
4
700
5
700
6
700
7
700
8
700
9
699
12
699
485
699
560
699
643
700
696

Status: Draft

onmental conditions

Page
699
1
699
1
699
540

CWE Version 1.4 Failure to Preserve SQL Query Structure (SQL Injection)

es.
Eliminate the SQL

Good Code

ge boxes, the
ation user has the

a database.

Bad Code

First of all, the
in SQL. If a user
which may bypass
data / command
able to alter the
possibly accessing
strophe are
a programmer may
prevent any data /

lows SQL injection

ly encoded output.
se Java Beans,
.

tion between
g, encoding, and
ability at every

tored procedures.
ng. Do not
"exec" or similar

CWE Version 1.4 Index

7 - Characters and

8 - Memory Management

9 - Input Output (FIO),

0 - Environment (ENV),

1 - Signals (SIG), 736

2 - Error Handling (ERR),

9 - Miscellaneous (MSC),

0 - POSIX (POS), 738

oint ('Man-in-the-Middle').

Comparison Errors, 200

ormation, 338

ve Information, 342

r-Side Security, 596

ne(), 578

le, 211

ossible Directory, 560

510

514

Instead of Object

r Modification of Security-

ar Buffers, 10

762

rors), 247

h, 396

actory with Incorrect

nsure Permissions,

nal Modifier, 521

750

ation Leak, 244

(F), 373

During Sensitive

or Injection'), 611

6

Exception, 424

Exception, 425

bl, 487

61

it Timing Channel, 539

CWE Version 1.4 Index

uplicate Identifier, 692

for Authorization

h to Detect NULL Pointer

ary Authentication, 334

erminate Size, 492

ction, 670

n, 333

-thread-safe Manner,

490

Argument, 564

Comparison, 592

79

ation, 319

bles, 651

is, 287

anism for Forgotten

541

ERT C Secure Coding

TE, 620

(2004), 718

(2007), 619

n C, 652

n C++, 653

n Java, 655

n PHP, 657

ANS Top 25 Most

739

esign, 696

plementation, 703

Directories, 621

622

Processes, 622

), 59

3

jection), 107

Index

CWE web site visitors by City



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

automation of vulnerability management, security measurement, and compliance (e.g. FISMA).

Resource Status

NVD contains:
26736 [CVE Vulnerabilities](#)
114 [Checklists](#)
91 [US-CERT Alerts](#)
1997 [US-CERT Vuln Notes](#)
2966 [OVAL Queries](#)
12410 [Vulnerable Products](#)

Last updated: 09/26/07
CVE Publication rate: 16 vulnerabilities / day

Email List

Select the email list(s) you wish to join, enter your e-mail address and press "Add" to receive NVD announcements or SCAP information.

NVD Announcements
 SCAP Announcements
 SCAP Discussion List
 XCCDF Discussion List

Workload Index

Vulnerability Workload Index: 9.06

About Us

NVD is a product of the NIST [Computer Security Division](#) and is sponsored by the Department of Homeland Security's [National Cyber Security Division](#). It supports the

Overview

SQL injection vulnerability in mods/banners/navlist.php in Clansphere 2007.4 allows remote attackers to execute arbitrary SQL commands via the cat_id parameter to index.php in a banners action.

Impact

CVSS Severity (version 2.0):
CVSS v2 Base score: 7.5 (High) (AV:N/AC:L/Au:N/C:P/I:P/A:P) (legend)
Impact Subscore: 6.4
Exploitability Subscore: 10.0

Access Vector: Network exploitable
Access Complexity: Low
Authentication: Not required to exploit
Impact Type: Provides unauthorized access, Allows partial confidentiality, integrity, and availability violation, Allows unauthorized disclosure of information, Allows disruption of service

References to Advisories, Solutions, and Tools

External Source: BID ([disclaimer](#))
Name: 25770
Hyperlink: <http://www.securityfocus.com/bid/25770>

External Source: MILWORM ([disclaimer](#))
Name: 4443
Hyperlink: <http://www.milw0rm.com/exploits/4443>

Vulnerable software and versions

Configuration 1
- Clansphere, Clansphere, 2007.4

Technical Details

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

CVE Standard Vulnerability Entry:
<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5061>

Common Platform Enumeration:

NVD XML feeds also include CWE

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

CWE Common Weakness Enumeration

A Community-Developed Dictionary of Software Weakness Types

Home > CWE List > CWE-89 Individual Dictionary Definition (Draft 9) [View the CWE List](#)

CWE-89 Individual Dictionary Definition (Draft 9)

Failure to Sanitize Data into SQL Queries (aka 'SQL Injection')

Weakness ID: 89 (Weakness Base) **Status:** Incomplete

Description: **Summary**
The application fails to adequately filter SQL syntax from user-controllable input. This can lead to such input being interpreted as SQL, rather than ordinary user data and be executed as part of a dynamically generated SQL query. This is a specific form of an injection problem, one that explicitly affects SQL databases, in which SQL commands are injected into data-plane input in order to effect the execution of dynamically generated SQL statements.

Likelihood of Exploit: Very High

Common Consequences:
Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.
Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.
Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability.
Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.

Potential Mitigations:
Requirements specification: A non-SQL style database which is not subject to this flaw may be chosen.
Design: Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data.
Implementation: Duplicate any filtering done on the client-side on the server side.
Implementation: Implement SQL strings using prepared statements that bind variables. Prepared statements that do not bind variables can be vulnerable to attack.

Section Contents
CWE List
Full Dictionary View
Classification Tree
Reports
Other Items of Interest
Sources
Key
- Weakness
- Base
- Variant
- Class
- Chain
- Composite
- Category
- View
- Deprecated

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-2008-4844](#).

Before I get started, I want to explain the goals of the SDL and the security work here at Microsoft. The SDL is designed as a multi-layered process to help systemically reduce security vulnerabilities; if one component of the SDL process fails to prevent or catch a bug, then some other component should prevent or catch the bug. The SDL also mandates the use of security defenses whose impact will be reflected in the "mitigations" section of a security bulletin, because we know that no software development process will catch all security bugs. As we have said many times, the goal of the SDL is to "Reduce vulnerabilities, and reduce the severity of what's missed."

In this post, I want to focus on the SDL-required code analysis, code review, fuzzing and compiler and operating system defenses and how they fared.

Background

The bug was an invalid pointer dereference in MSHTML.DLL when the code handles data binding. It's important to point out that there is no heap corruption and there is no heap-based buffer overrun!

When data binding is used, IE creates an object which contains an array of data binding objects. In the code in question, when a data binding object is released, the array length is not correctly updated leading to a function call into freed memory.

The vulnerable code looks a little like this (by the way, the real array name is `_aryPXfer`, but I figured `ArrayOfObjectsFromIE` is a little more descriptive for people not in the Internet Explorer team.)

```
int MaxIdx = ArrayOfObjectsFromIE.Size()-1;
for (int i=0; i <= MaxIdx; i++) {
    if (!ArrayOfObjectsFromIE[i])
        continue;
    ArrayOfObjectsFromIE[i]->TransferFromSource();
    ...
}
```

Here's how the vulnerability manifests itself: if there are two data transfers with the same identifier (so `MaxIdx` is 2), and the first transfer updates the length of the `ArrayOfObjectsFromIE` array when its work was done and releases its data binding object, the loop count would still be whatever `MaxIdx` was at the start of the loop, 2.

This is a time-of-check-time-of-use (TOCTOU) bug that led to code calling into a freed memory block. The Common Weakness Enumeration (CWE) classification for this vulnerability is [CWE-367](#).

The fix was to check the maximum iteration count on each loop iteration rather than once before the loop starts; this is the correct fix for a TOCTOU bug - move the check as close as possible to the action because in

a time-of-check-time-of-use (TOCTOU) bug that led to code calling into a freed memory block. The 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

SAMATE Reference Dataset

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

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

» sign in register | Search... GO

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

Welcome to the NIST SAMATE Reference Dataset Project

The purpose of the SAMATE Reference Dataset (SRD) is to provide users, researchers, and software security assurance tool developers with a set of known security flaws. This will allow end users to evaluate tools and tool developers to test their methods. These test cases are designs, source code, binaries, etc., i.e. from all the phases of the software life cycle. The dataset includes "wild" (production), "synthetic" (written to test or generated), and "academic" (from students) test cases. This database will also contain real software applications with known bugs and vulnerabilities. The dataset intends to encompass a wide variety of possible vulnerabilities, languages, platforms, compilers. The dataset is anticipated to become a large-scale effort, gathering test cases from many contributors. We have more information 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 browse the test case repository; or 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

NIST Special Publications:

- SP500-268 CWE
- SP500-269 CWE
- SP800-53a CVE, OVAL, CWE
- SP800-115 CVE, CCE, CVSS, CWE

NIST Interagency Reports:

- NISTIR-7435 CVE, CVSS, CWE
- NISTIR-7628 CVE, CWE



U.S. Department of Energy
Office of Electricity Delivery
and Energy Reliability

INL/EXT-10-18381

NSTB Assessments Summary Report: Common Industrial Control System Cyber Security Weaknesses

May 2010

NSTB

National SCADA Test Bed
Enhancing control systems security in the energy sector



Idaho National Labs SCADA Report

SECURE CONTROL SYSTEM/ENTERPRISE ARCHITECTURE

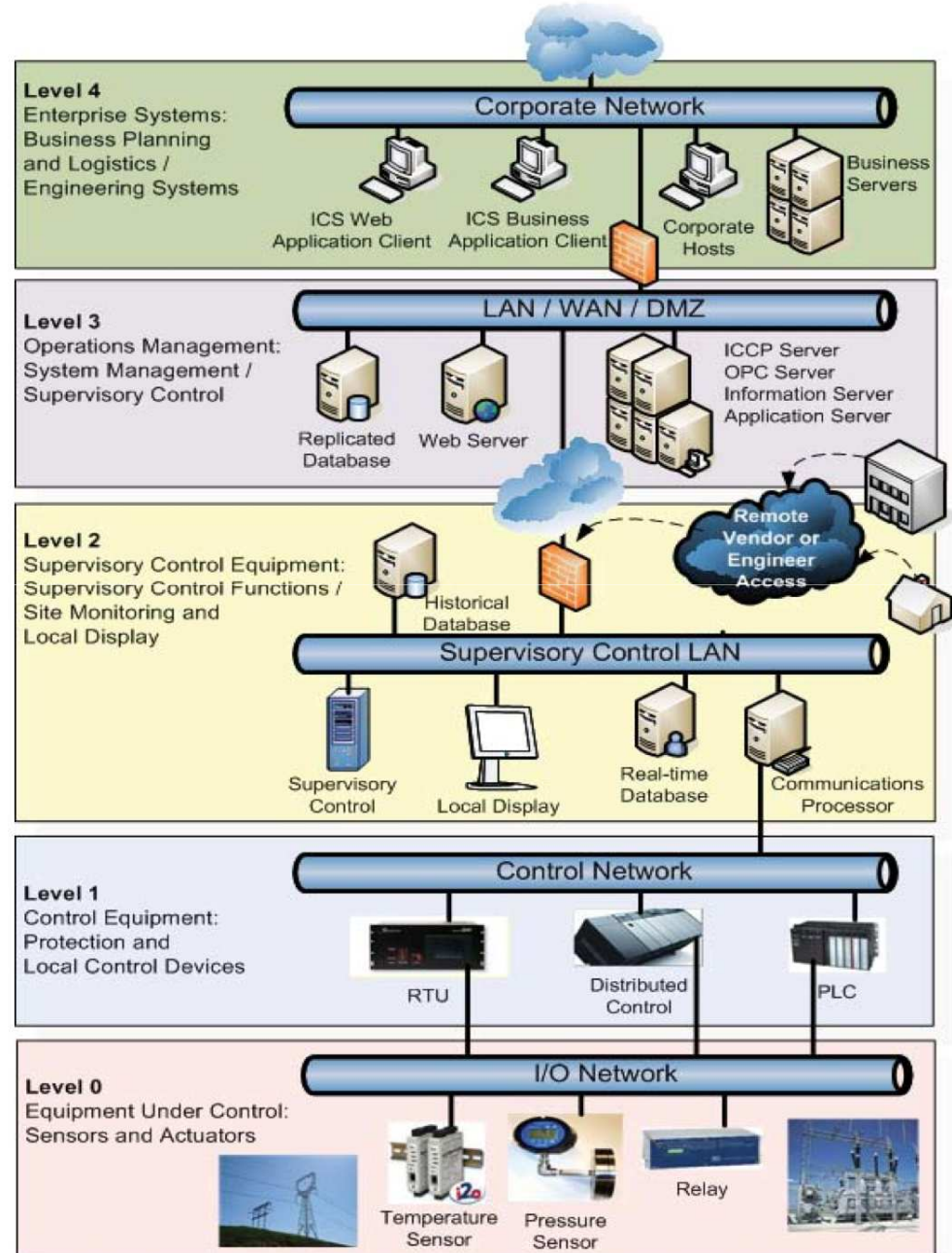
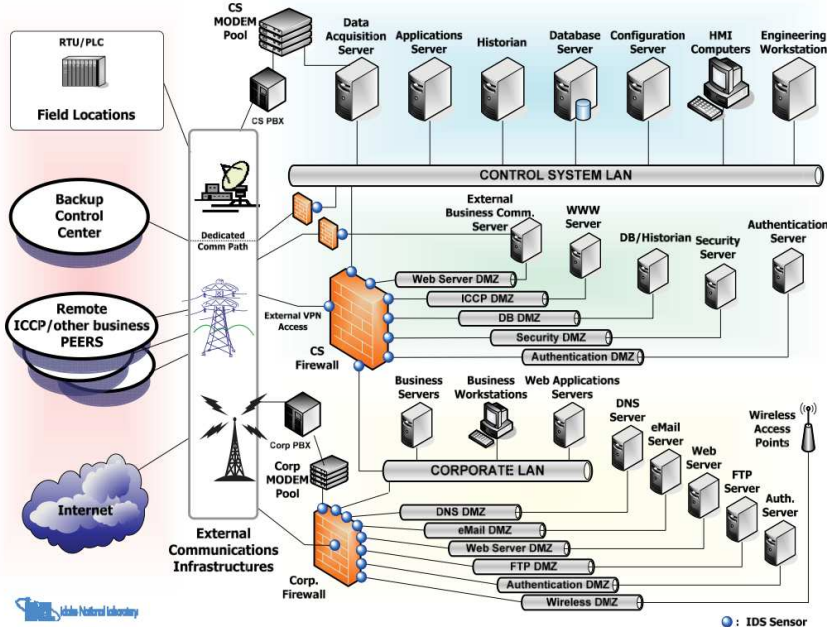


Table 27. Most common programming errors found in ICS code.

Weakness Classification	Vulnerability Type
CWE-19: Data Handling	CWE-228: Improper Handling of Syntactically Invalid Structure
	CWE-229: Improper Handling of Values
	CWE-230: Improper Handling of Missing Values
	CWE-20: Improper Input Validation
	CWE-116: Improper Encoding or Escaping of Output
	CWE-195: Signed to Unsigned Conversion Error
	CWE-198: Use of Incorrect Byte Ordering
CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer	CWE-120: Buffer Copy without Checking Size of Input (“Classic Buffer Overflow”)
	CWE-121: Stack-based Buffer Overflow
	CWE-122: Heap-based Buffer Overflow
	CWE-125: Out-of-bounds Read
	CWE-129: Improper Validation of Array Index
	CWE-131: Incorrect Calculation of Buffer Size
	CWE-170: Improper Null Termination
	CWE-190: Integer Overflow or Wraparound
	CWE-680: Integer Overflow to Buffer Overflow
CWE-398: Indicator of Poor Code Quality	CWE-454: External Initialization of Trusted Variables or Data Stores
	CWE-456: Missing Initialization
	CWE-457: Use of Uninitialized Variable
	CWE-476: NULL Pointer Dereference
	CWE-400: Uncontrolled Resource Consumption (“Resource Exhaustion”)
	CWE-252: Unchecked Return Value
	CWE-690: Unchecked Return Value to NULL Pointer Dereference
	CWE-772: Missing Release of Resource after Effective Lifetime
CWE-442: Web Problems	CWE-22: Improper Limitation of a Pathname to a Restricted Directory (“Path Traversal”)
	CWE-79: Failure to Preserve Web Page Structure (“Cross-site Scripting”)
	CWE-89: Failure to Preserve SQL Query Structure (“SQL Injection”)
CWE-703: Failure to Handle Exceptional Conditions	CWE-431: Missing Handler
	CWE-248: Uncaught Exception
	CWE-755: Improper Handling of Exceptional Conditions
	CWE-390: Detection of Error Condition Without Action

16 July 2010

A Human Capital Crisis in Cybersecurity

Technical Proficiency Matters

A White Paper of the
CSIS Commission on Cybersecurity for the 44th Presidency

COCHAIRS
Representative James R. Langevin
Representative Michael T. McCaul
Scott Charney
Lt. General Harry Raduege
USAF (ret.)

PROJECT DIRECTOR

J

A complete body of knowledge covering the entire field of software engineering may be years away. However, the body of knowledge needed by professionals to create software free of common and critical security flaws has been developed, vetted widely and kept up to date. That is the foundation for a certification program in software assurance that can gain wide adoption. It was created in late 2008 by a consortium of national experts, sponsored by DHS and NSA, and was updated in late 2009. It contains ranked lists of the most common errors, explanations of why the errors are dangerous, examples of those errors in multiple languages, and ways of eliminating those errors. It can be found at <http://cwe.mitre.org/top25>.

Any programmer who writes code without being aware of those problems and is not capable of writing code free of those errors is a threat to his or her employers and to others who use computers connected to systems running his or her software.

based on a body of knowledge that represents the complete set of concepts, terms and activities that make up a professional domain. And absent such a body of knowledge there is little basis for supporting a certification program. Indeed it would be dangerous and misleading.

A complete body of knowledge covering the entire field of software engineering may be years away. However, the body of knowledge needed by professionals to create software free of common and critical security flaws has been developed, vetted widely and kept up to date. That is the foundation for a certification program in software assurance that can gain wide adoption. It was created in late 2008 by a consortium of national experts, sponsored by DHS and NSA, and was updated in late 2009. It contains ranked lists of the most common errors, explanations of why the errors are dangerous, examples of those errors in multiple languages, and ways of eliminating those errors. It can be found at <http://cwe.mitre.org/top25>.

Any programmer who writes code without being aware of those problems and is not capable of writing code free of those errors is a threat to his or her employers and to others who use computers connected to systems running his or her software.

Industry Uptake

Foreword

In 2008, the Software Assurance Forum for Excellence in Code (SAFECode) published the first version of this report in an effort to help others in the industry initiate or improve their own software assurance programs and encourage the industry-wide adoption of what we believe to be the most fundamental secure development methods. This work remains our most in-demand paper and has been downloaded more than 50,000 times since its original release.

However, secure software development is not only a goal, it is also a process. In the nearly two and a half years since we first released this paper, the process of building secure software has continued to evolve and improve alongside innovations and advancements in the information and communications technology industry. Much has been learned not only through increased community collaboration, but also through the ongoing internal efforts of SAFECode's member companies. This 2nd Edition aims to help disseminate that new knowledge.

Just as with the original paper, this paper is not meant to be a comprehensive guide to all possible secure development practices. Rather, it is meant to provide a foundational set of secure development practices that have been effective in improving software security in real-world implementations by SAFECode members across their diverse development environments.

It is important to note that these are the "practiced practices" employed by SAFECode members, which we identified through an ongoing analysis of our members' individual software security efforts. By

bringing these methods together and sharing them with the larger community, SAFECode hopes to move the industry beyond defining theoretical best practices to describing sets of software engineering practices that have been shown to improve the security of software and are currently in use at leading software companies. Using this approach enables SAFECode to encourage the adoption of best practices that are proven, repeatable, and implementable even when requirements and development processes are in flux.

Though expanded, our key goals remain—keep it concise, actionable, and focused.

What's New

This edition of the paper prescribes updated security practices that span the entire software development lifecycle, from Design, Programming, and Testing to Deployment, Operations, and Maintenance. Practices have been shown to be effective in diverse development environments, from original development to maintenance and updates. The paper also covers Training, Remediation, and Documentation, topics that have not been given detailed treatment in previous editions. The paper also covers security engineering training and software integrity in the global supply chain, and thus we have refined our focus in this paper to concentrate on the core areas of design, development and testing.

The paper also contains two important, additional sections for each listed practice that will further increase its value to implementers—Common Weakness Enumeration (CWE) references and Verification guidance.



The paper also contains two important, additional sections for each listed practice that will further increase its value to implementers—Common Weakness Enumeration (CWE) references and Verification guidance.



Verification plan is a derivative of the results of the Threat Model activity. The Threat Model itself will serve as a clear road map for verification, containing enough information to identify each threat and mitigation that can be verified. During verification, the Threat Model and mitigated threats, as well as the annotated architectural diagrams, should also be made available to testers in order to help define further test cases and refine the verification process. A review of the Threat Model and verification results should be made an integral part of the activities required to declare code complete.

CWE References

Much of CWE focuses on implementation issues, and Threat Modeling is a design-time event. There are, however, a number of CWEs that are applicable to the threat modeling process, including:

- CWE-287: Improper authentication is an example of weakness that could be exploited by a Spoofing threat
- CWE-264: Permissions, Privileges, and Access Controls is a parent weakness of many Tampering, Repudiation and Elevation of Privilege threats
- CWE-311: Missing Encryption of Sensitive Data is an example of an Information Disclosure threat
- CWE-400: (uncontrolled resource consumption) is one example of an unmitigated Denial of Service threat

An example of a portion of a test plan derived from a Threat Model could be:

Threat Identified	Design Element(s)	Mitigation	Verification
Session Hijacking	GUI	Ensure random session identifiers of appropriate length	Collect session identifiers over a number of sessions and examine distribution and length
Tampering with data in transit	Process A on server to Process B on client	Use SSL to ensure that data isn't modified in transit	Assert that communication cannot be established without the use of SSL

CWE

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

Fundamental Practices for Secure Software Development
2ND EDITION

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

February 8, 2011

Editor: Stacy Simpson, SAFECode

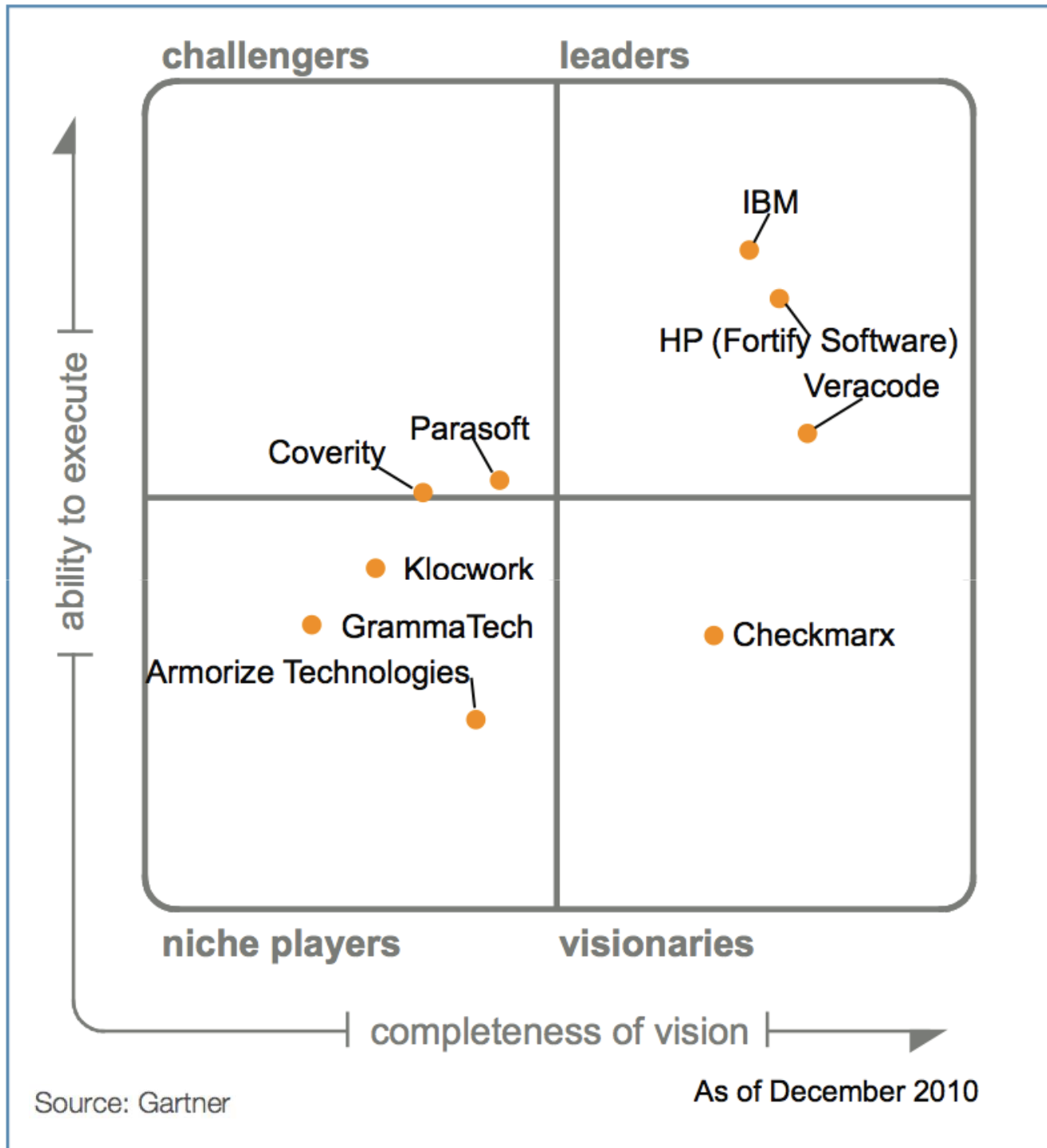
AUTHORS
Mark Bell, Juniper Networks; Matt Coles, EMC Corporation; Cassio Goldschmidt, Symantec Corp.; Michael Howard, Microsoft Corp.; Kyle Randolph, Adobe Systems Inc.; Mikko Saario, Nokia; Reeny Sondhi, EMC Corporation; Izar Taranadach, EMC Corporation; Antti Vähä-Sipilä, Nokia; Yonko Yonchev, SAP AG



Gartner Magic Quadrant for Static Application Security Testing Tools

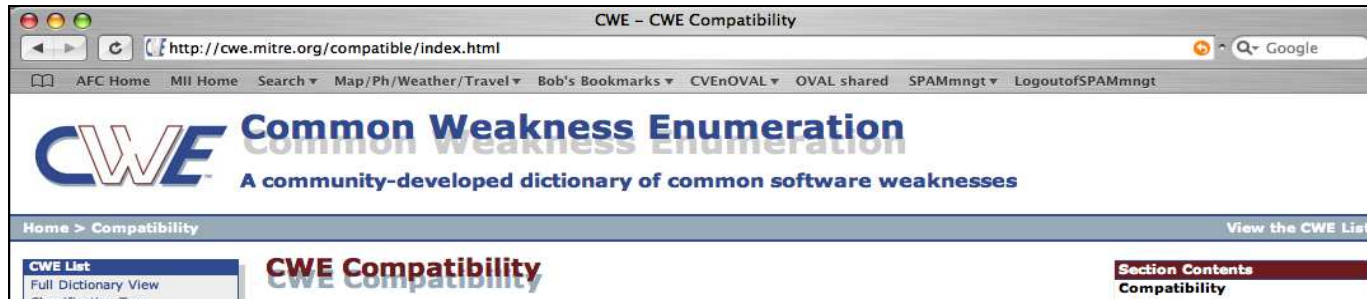
Plus Some Other Important Tool Players...

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



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.

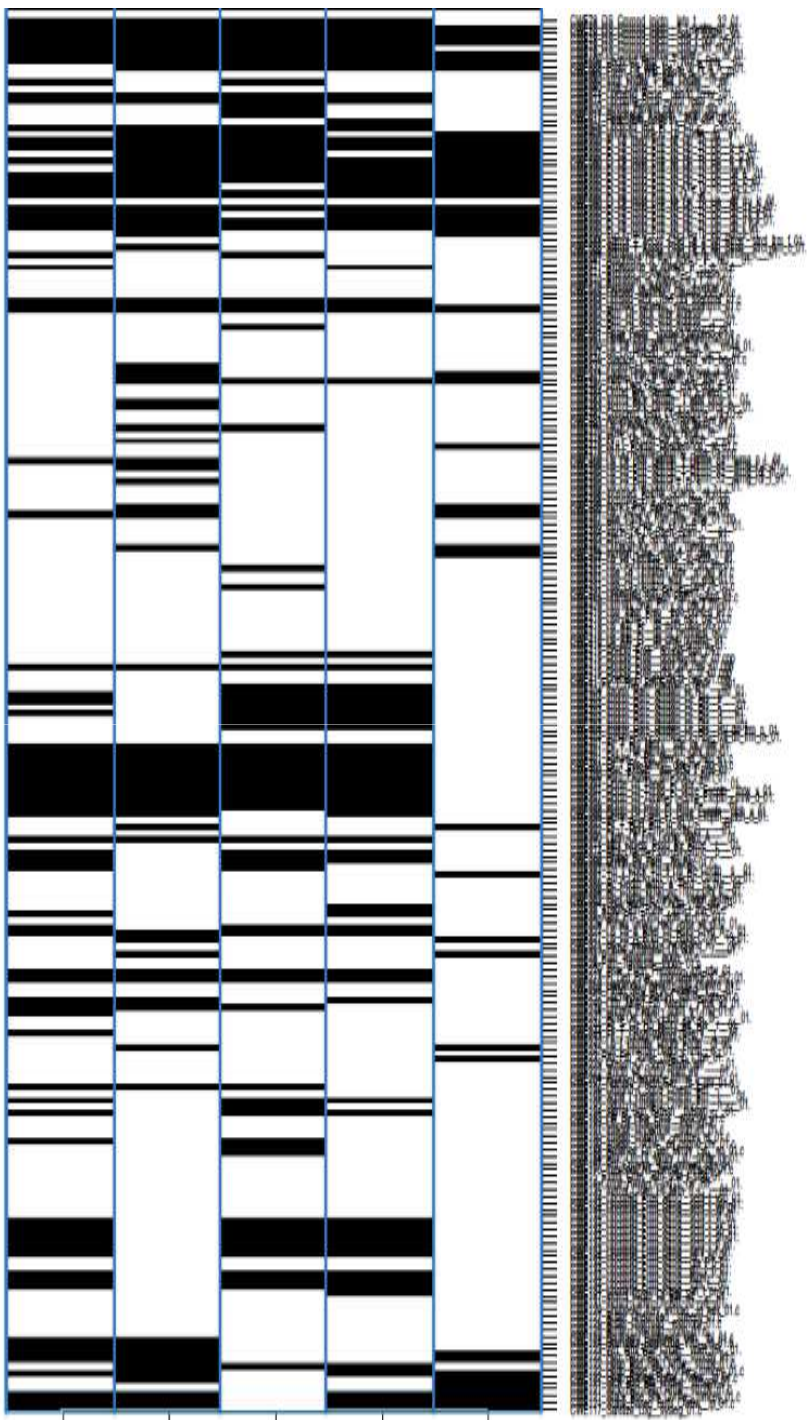
Products are listed alphabetically by organization name:

cwe.mitre.org/compatible/

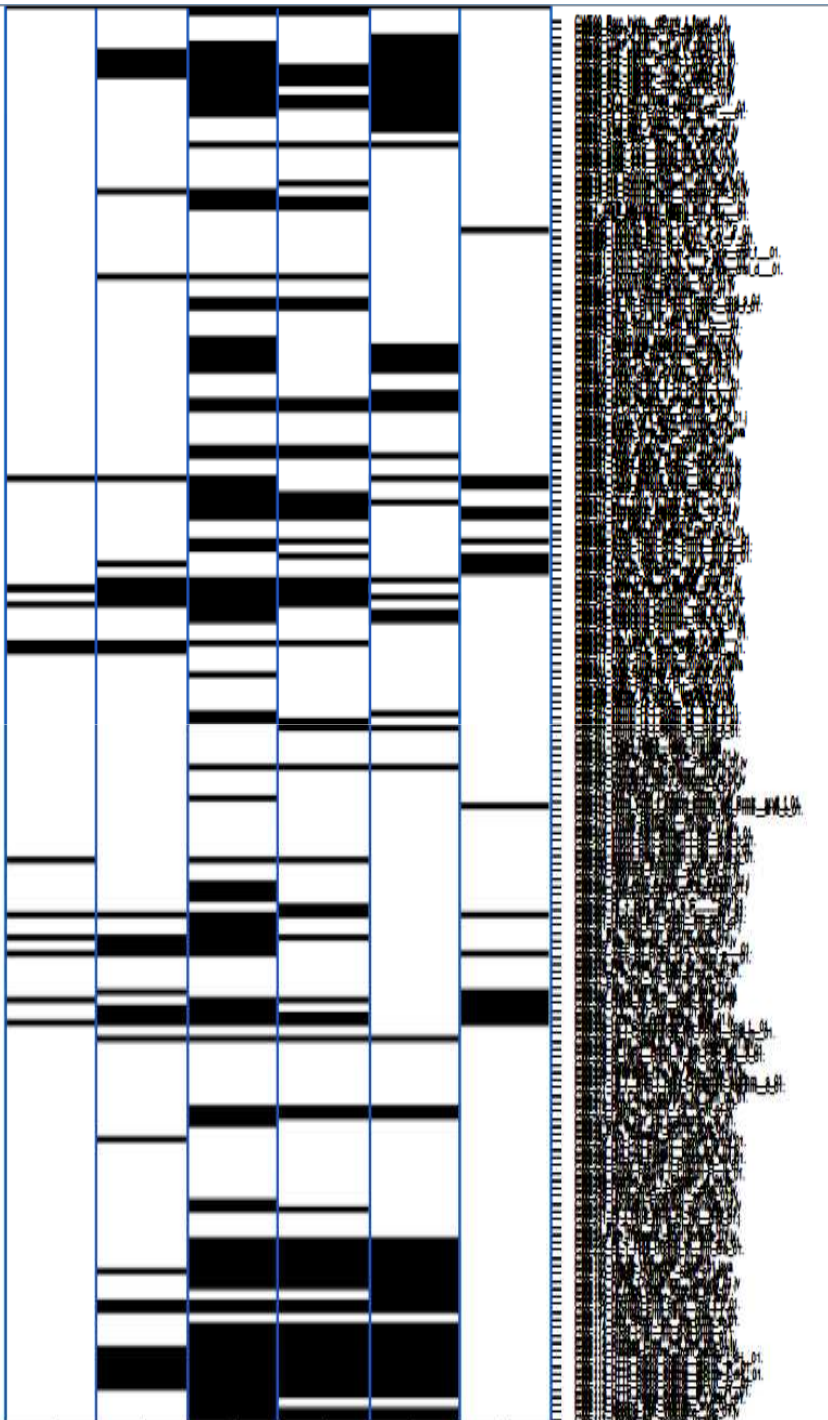
TOTALS
Organizations Participating: 29
Products & Services: 48

December 29, 2006

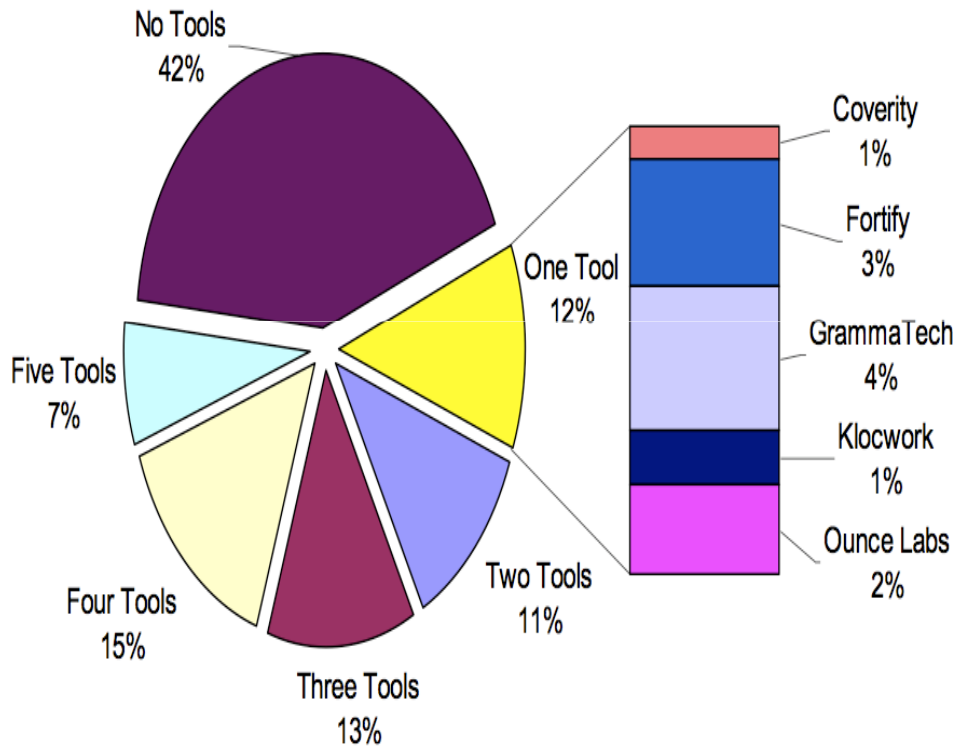
C Test Cases



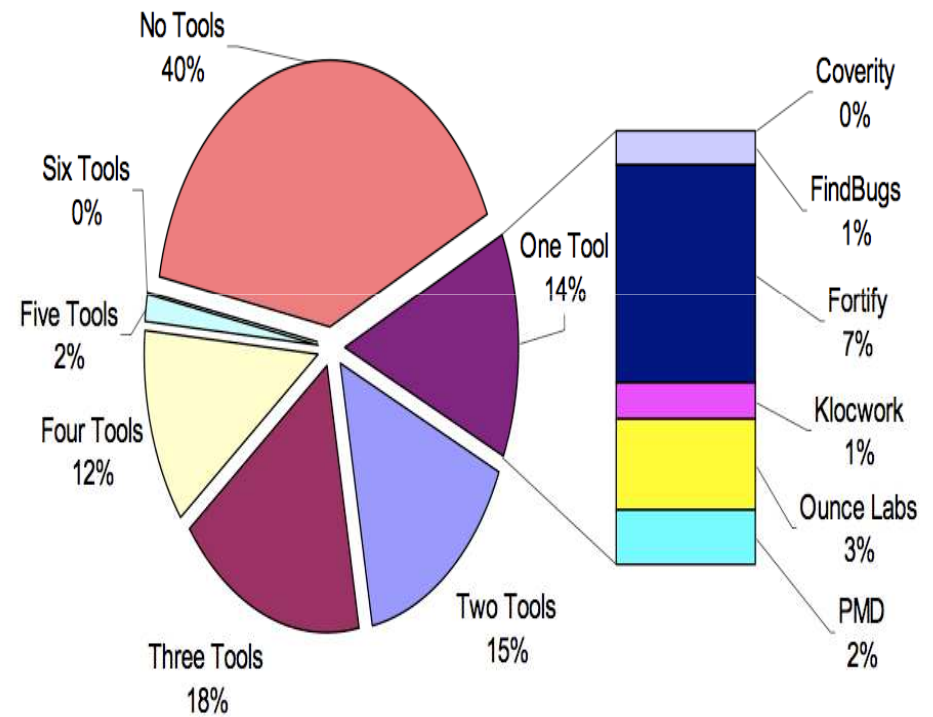
Java Test Cases



C/C++ "Breadth" Test Case Coverage



Java "Breadth" Test Case Coverage



Code Analysis Effectiveness Assessment...



CWE
Validation
Effectiveness
Testing - ?

CWE
Compatibility
and
Effectiveness

CWEs with
WhiteBox
Definitions

Center For
Assure SW
Tool Evaluation
2007
Tool Evaluation
2009

IARPA
STONESOUP-
Securely Taking
On New
Executable Stuff
Of Uncertain
Provenance

OSD/NII
CWE
Formal

NIST
SAMATE
SP 500-267
SP 500-269
SP 500-270

SAMATE
Repository
Dataset
(SRD)
Automated
Test Case
Generator

NIST SATE
SATE08
SATE09
SATE10

SySA Task
Force
WhiteBox
Definitions-to-
SBVR-to-
microKDM

All of these are aimed at different aspects of understanding how well tools find CWEs in software applications and what can be done to improve that and standardize the process for expressing a tools capabilities.

CWE Coverage – Implemented...

Coverity Data Sheet



Coverity Coverage for Common Weakness Enumeration (CWE): Java

CWE ID	Coverity Static Analysis Checker
171	BAD_EQ
252	CHECKED_RETURN
366	GUARDED BY VIOLATION
	INDIRECT GUARDED BY VIOLATION
	NON_STATIC_GUARDING_STATIC
	VOLATILE_ATOMICTY
382	DC.CODING_STYLE
398	BAD_OVERRIDE
	DC.EXPLICIT_DEPRECATION
	DC.GC
399	MUTABLE_COMPARISON
	MUTABLE_HASHCODE

Coverity Data Sheet



Coverity Coverage For Common Weakness Enumeration (CWE): C/C++

CWE ID	Coverity Static Analysis Checker	Checker Description	Type of Security Risk	
	TAINTED_SCALAR	Use of untrusted scalar value	Arbitrary code execution	
		Untrusted value as an argument		
		Use of untrusted value		
		Use of untrusted string value		
		User pointer dereference		
		Out-of-bounds access		
		Stray pointer arithmetic		
		COM bad conversion to BSTR		
		Overflowed array index write		
		Overflowed pointer write		
	TAINTED_SCALAR	Using invalid iterator	Arbitrary code execution	
		Iterator container mismatch		
		Splice iterator mismatch		
		Allocation size error		Denial of service
		Out-of-bounds access		
		Out-of-bounds write		
		Out-of-bounds access		
		Out-of-bounds write		
		Argument cannot be negative		
		Copy into fixed size buffer		
Destination buffer too small				
Possible buffer overflow				
Allocation too small for type	Unauthorized code execution			
Buffer overflow				
Copy into fixed size buffer				
Destination buffer too small				
Unbounded source buffer				

CWE IDs mapped to Klocwork Java issue types - current

<http://www.klocwork.com/products/documentation/current...>

CWE IDs mapped to Klocwork Java issue types

From current

CWE IDs mapped to Klocwork Java issue types

See also Detected Java Issues.

CWE IDs mapped to Klocwork C and C++ issue types/ja - ...

<http://www.klocwork.com/products/documentation/current...>

CWE IDs mapped to Klocwork C and C++ issue types/ja

From current

< CWE IDs mapped to Klocwork C and C++ issue types

CWE IDs mapped to Klocwork C and C++ issue types/ja

その他の情報 Detected C and C++ Issues.

CWE ID	説明
20 (http://cwe.mitre.org/data/definitions/20.html)	ABV.IAIN.LBU 未検証入力によるバッファ オーバーフロー SV.TAINTED.GENERIC 未検証文字列データの使用 SV.TAINTED.ALLOC_SIZE メモリ割り当てにおける未検証の整数の使用 SV.TAINTED.CALL_INDEX_ACCESS 関数呼び出しにおける未検証整数の配列インデックスとしての使用
22 (http://cwe.mitre.org/data/definitions/22.html)	SV.CUDS.MISSING_ABSOLUTE_PATH ファイルのロードでの絶対パスの不使用
73 (http://cwe.mitre.org/data/definitions/73.html)	SV.CUDS.MISSING_ABSOLUTE_PATH ファイルのロードでの絶対パスの不使用
74 (http://cwe.mitre.org/data/definitions/74.html)	SV.TAINTED.INJECTION コマンド インジェクション
77 (http://cwe.mitre.org/data/definitions/77.html)	SV.CODE_INJECTION.SHELL_EXEC シェル実行へのコマンド インジェクション
78 (http://cwe.mitre.org/data/definitions/78.html)	NNTS.TAINTED 未検証ユーザ入力がある原因のバッファ オーバーフロー - 非 NULL 終端文字列 SV.TAINTED.INJECTION コマンド インジェクション
88 (http://cwe.mitre.org)	SV.TAINTED.INJECTION コマンド インジェクション NNTS.TAINTED 未検証ユーザ入力がある原因のバッファ オーバーフロー

10/7

2/28/11 10:34 AM



www.cenzic.com | (866) 4-CENZIC (866-423-8842)

Cenzic Product Suite is CWE Compatible

Cenzic HalStorm Enterprise ARC, Cenzic HalStorm Professional and Cenzic ClickToSecure are compatible with the CWE standard or Common Weakness Enumeration as maintained by Mitre Corporation. Web security assessment results from the HalStorm product suite are mapped to the relevant CWE ID's providing users with additional information to classify and describe common weaknesses found in Web applications.

For additional details on CWE, please visit: <http://www.mitre.org/cwe.html>

The following is a mapping between Cenzic's SmartAttack and CWE ID's:

Cenzic SmartAttack Name	CWE ID's
1 Application Exception	CWE-398: Error Handling
2 Application Exception (WS)	CWE-398: Error Handling
3 Application Path Disclosure	CWE-200: Information Leak (rough match)
4 Authentication Bypass	CWE-89: Failure to Sanitize Data into SQL Queries (aka 'SQL Injection') (rough match)
5 Authorization Boundary	CWE-286: Missing or Inconsistent Access Control. CWE-426: Direct Request ('Forced Browsing')
6 Blind SQL Injection	CWE-89: Failure to Sanitize Data into SQL Queries (aka 'SQL Injection')
7 Blind SQL Injection (WS)	CWE-89: Failure to Sanitize Data into SQL Queries (aka 'SQL Injection')
8 Browse HTTP from HTTPS List	CWE-200: Information Leak
9 Brute Force Login	CWE-521: Weak Password Requirements
10 Buffer Overflow	CWE-120: Unbounded Transfer ('Classic Buffer Overflow')
11 Buffer Overflow (WS)	CWE-120: Unbounded Transfer ('Classic Buffer Overflow')
12 Check Basic Auth over HTTP	CWE-200: Information Leak
13 Check HTTP Methods	CWE-850: Trusting HTTP Permission Methods on the Server Side

Cenzic CWE Structure | October 2009

3

Copyright © 2009 MITRE Corporation. All rights reserved. This document is the property of MITRE Corporation. All other rights reserved.

Special thanks to Robert A. Wierba of MITRE Corporation.

Handler Errors

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

User Interface Errors

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

Behavioral Problems

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

Initialization and Cleanup Errors

- Insecure Default Variable Initialization
- External Initialization of Trusted Variable
- Run-ask on Failed Initialization
- Missing Initialization
- Incomplete Cleanup
- Improper Cleanup on Threat Exception
- Improper Initialization (149)

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 Standard Error Handling Mechanism
- Failure to Catch All Exceptions in Servlet
- Not Falling Securely (Falling Open)
- Missing Custom Error Page

Pointer Issues

- Return of Pointer Value Outside of Expected Range
- Use of size off 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 Definition
 - State Synchronization on Error
 - Mutable Objects Passed by Reference
 - Passing Mutable Object to an Unchecked Method
 - External Control of Critical State Data (348)
- Race Conditions (292)
- Session Fixation
- Concurrency Issues
- Temporary File Issues
- Covert Timing Channel
- Technology-Specific Time and State Issues
- Synthetic Name not Mapping to Correct Object
- Signal Errors
- 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 (364)
- Empty Synchronized Block
- Explicit Call to Finalize
- Reachable Assertion
- Use of Potentially Dangerous Function

Failure to Fulfill API Contract ('API Abuse')

- Failure to Clear Heap Memory Before Release (Heap Inspection)
- Call to Non-abstract API
- Use of Inherently Dangerous Function
- Multiple Binds to the Same Port
- J2EE Bad Practices: Direct Management of Connections
- Incorrect Check of Function Return Value
- Open Missing Arguments and Parameters
- Uncaught Exception
- Assertion with Unnecessary Privileges (230)
- Open Missing String Management
- J2EE Bad Practices: Direct Use of Sockets
- Unchecked Return Value
- Failure to Change Working Directory in chroot jail
- Reliance on DNS Lookups in chroot jail
- Failure to Follow Specification
- Failure to Provide Specified Functionality

Web Problems

- Failure to Sanitize CGI 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 (364)
- Empty Synchronized Block
- Explicit Call to Finalize
- Reachable Assertion
- Use of Potentially Dangerous Function

Credentials Management

- Hard-Coded Password (288)
- Unverified Password Change
- Missing Password Field Masking
- Weak Cryptography for Passwords
- Weak Password Requirements
- Not Using Password Aging
- Password Aging with Long Expiration
- Insecurely Protected Credentials
- Weak Password Recovery Mechanism or Forgotten Password

Insufficient Verification of Data Authenticity

- Single Validation Error
- Improper Verification of Cryptographic Signature
- Use of Low-Trust Source
- Acceptance of Extraneous Untrusted Data With Trained Data
- Improperly Trusted Remote DNS
- Insecure Type Detection
- Open Missing Parsers (395) (333)
- 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-Dependent File
- Reliance on Obfuscation or Obfuscation of Security-Related Inputs without Integrity Checking

Privacy Violation

- Reliance on Cookies without Validation and Integrity Checking
- Client Side Enforcement of Server Side Security (342)
- Improperly Implemented Security Check for Standard
- Improper Authentication
- User Interface Security Issues
- Use of Insufficiently Random Values (330)
- Logging of Excessive Data
- Certainty Issues

Insufficient Encapsulation

- Mobile Code Issues/Missing Custom Error Page
 - Public (Accessible) Method Without Final (Object Back)
 - Use of Inner Class Containing Sensitive Data
 - Critical Public Variable Without Final Modifier
 - Serialized Object Without Integrity Check (346)
 - Array Declared Public, Final, and Static
 - Finalize() Method Declared Public
- Leftover Debug Code
- Use of Dynamic Class Loading
 - clone() Method Without super.clone()
 - Comparison of Classes by Name
- Data Leak Between Sessions
- Trust Boundary Violation

Cryptographic Issues

- Key Management Errors
 - Missing Required Cryptographic Step
 - Not Using a Random IV with CBC Mode
 - Failure to Encrypt Sensitive Data
 - Incorrect Storage of Encryption Information
 - Clearable Transmission of Sensitive Information (316)
 - Sensitive Code in a HTTPS Session Without Secure Attributes
 - Removable One-Time Pad
 - Inadequate Encryption Strength
 - Use of a Broken or Flawed Cryptographic Algorithm (327)
 - Use of RSA Algorithms without OAEP

Permissions, Privileges, and Access Controls

- Access Control Mechanism Issues (248)
 - Permission Issues
 - Incorrect Default Permissions
 - Incorrect Inherited Permissions
 - Incorrectly Presumed Inherited Permissions
 - Incorrect Execution Assigned Permissions
 - Improper Handling of Exec/Exec Permissions
 - Improper Permission of Permissions
 - Exposed Usually Access Method
 - Incorrect Permissions Assigned for Critical Resources (234)
 - Permission Race Condition During Resource Copy
 - Privilege / Session Issues
 - Improper Ownership Management
 - Insecure User Management

Password in Configuration File

- Insufficient Compartmentalization
- Reliance on a Single Factor in a Security Decision
- Insufficient Psychological Acceptability
- Reliance on Security through Obscurity
- Protection Mechanism Failure
- Insufficient Logging
- Reliance on Cookies without Validation and Integrity Checking in a Security Decision

Data Handling

- Numeric Errors
 - Use of Incorrect Byte Ordering
 - Unchecked Array Indexing
 - Incorrect Conversion Between Numeric Types
 - Unchecked Sign Extension
 - Signed to Unsigned Conversion Error
 - Unsigned to Signed Conversion Error
 - Integer Truncation Error
 - Insecure Calculations (382)
 - Incorrect Calculation of Buffer Size
 - Integer Overflow or Wraparound
 - Integer Underflow (Wrap or Wraparound)
 - Overflow Error
 - Divide By Zero
- Modification of Assumed-Immutable Data (MAID)
 - Improper Input Validation (201)
 - Pathnames Traversal and Escape Sequence Errors
 - Process Control
 - Missing EM, Validation
 - Failure to Sanitize Data into a Different Plane (Injection)
 - Improper Sanitization of Special Elements used in a Comment (Comment Injection) (17)
 - Improper Sanitization of Special Elements used in an XML Comment (XML Injection) (84)
 - Failure to Sanitize Data into LAMP Queries (LAMP Injection)
 - XML Injection (aka Blind XPath Injection)
 - Failure to Sanitize CGI Parameters (CGI Injection)
 - Uncontrolled Format String
 - Failure to Sanitize Special Elements into a Different Plane
 - Argument Injection or Modification
 - Improper Control of Resource Methods (Resource Injection)
 - Failure to Control Retention of Cols (Track Injection) (36)
 - Improper Sanitization of Special Elements
 - Technology Specific Input Validation Problems
 - Maintainability of Input
 - Unchecked Input for Loop Condition
 - Null Byte Injection Error (Phone Null Byte)
 - Direct Use of Unicode JNI
 - Improper Output Sanitization for Log
 - Failure to Control Operator or within the Bounds of a Member Buffer (118)
 - Use of Externally-Controlled Input to Select Class or Code (Junk's Reflection)
 - ASP.NET Misconfigurations: Not Using Input Validation Framework
 - URL Redirection to Untrusted Site (Open Redirect)
 - Variable Extraction Errors
 - Unvalidated Function Hook Arguments
 - External Control of File Name or Path (71)
 - Improper Address Initialization (IDC), with METHOD, METHOD, IO Control Code
 - Use of Path Manipulation Function with Maximum-sized Buffer
- Representation Errors
 - Checking Generalizability, and Comparison Errors
 - Reliance on Data Memory Layout
- Information Management Errors
 - Information Leak (Information Disclosure)
 - Information Leak Through Text Data
 - Privacy Leak Through Data Objects
 - Discourage Information Leak
 - Open Missing Information Leak (336)
 - Open Arbitrary Channel Information Leak
 - Untrusted Information Leak
 - Process Environment Information Leak
 - Information Leak Through Debug Information
 - Service Information Disclosure Before Release
 - Information Leak of System Data
 - Information Leak Through Caching
 - Information Leak Through Environmental Variables
 - File and Directory Information Leak
 - Information Leak Through Query Strings in CGI Request
 - Information Leak Through Logging of Private Data
 - Information Leak in Decision
 - Contention Errors/Contention Error
 - Improper Access of Indexable Resource (Range Error)
 - Type Errors
 - Improper Casting or Casting of Output (114)
 - String Errors
 - Data Structure Issues
 - Improper Handling of Structurally Invalid Structure

With all of these CWEs, where do you start?

2009 SANS/CWE Top 25 Programming Errors

(released 12 Jan 2009) cwe.mitre.org/top25/

The image shows two overlapping browser windows. The top window is the SANS Institute website, titled "SANS Institute - CWE/SANS TOP 25 Most Dangerous Programming Errors". It features a navigation bar with buttons for "why SANS?", "pick a course", "why certify?", "register now", and a search box. Below the navigation bar is a banner with the SANS logo and the tagline "The right security training for your staff, at the right time, in the right place." The bottom window is the CWE website, titled "CWE - 2009 CWE/SANS Top 25 Most Dangerous Programming Errors". It features the CWE logo and the tagline "A Community-Developed Dictionary of Software Weakness Types". The main content area is titled "2009 CWE/SANS Top 25 Most Dangerous Programming Errors" and includes a "Document version: 1.0 (pdf)" link, a "Date: January 12, 2009", and "Project Coordinators: Bob Martin (MITRE), Mason Brown (SANS), Alan Paller (SANS)". A "Section Contents" sidebar lists "CWE/SANS Top 25", "Supporting Quotes", "Contributors", "On the Cusp", "Top 25 FAQ", "Top 25 Process", and "Change Log". The main text begins with "The 2009 CWE/SANS Top 25 Most Dangerous Programming Errors is a list of the most significant programming errors that can lead to serious software vulnerabilities. They occur frequently, are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all."

Making Security Measurable™

2010 CWE/SANS Top 25 Programming Errors

(released 16 Feb 2010) cwe.mitre.org/top25/

- Sponsored by:
 - National Cyber Security Division (DHS)
- List was selected by a group of security experts from 34 organizations including:
 - Academia: Purdue, Northern Kentucky University
 - Government: CERT, NSA, DHS
 - Software Vendors: Microsoft, Oracle, Red Hat, Apple, Juniper, McAfee, Symantec, Sun, RSA (of EMC)
 - Security Vendors: Veracode, Fortify, Mandiant, Cigital, SRI, Secunia, Breach, SAIC, Aspect,
 - Security Groups: OWASP, WASC

Making
Security
Measurable™

The screenshot shows the SANS website header with navigation links: training, certification, resources, vendor, portal, storm center, college, developer, about. The main content area is titled "CWE/SANS TOP 25 Most Dangerous Programming Errors" and features a banner for "SANS FIRE 2010 Baltimore, MD June 6-14". Below the banner, there is a section titled "What Errors Are Included in the Top 25 Programming Errors?" with a version number and update date. A list of error categories is provided, including "Insecure Interaction Between Components (8 errors)", "Risky Resource Management (10 errors)", and "Porous Defenses (7 errors)". A "Click Here" link is present. The page also includes a "Yearly Archive" section with links for 2010 and 2009. On the right side, there is a sidebar with a "SANS AppSec Streetfighter Blog" announcement and a "Real Threats, Real Skills, Real Success" banner for the SANS Cyber Guardian Program.

Main Goals

- Raise awareness for developers
- Help universities to teach secure coding
- Empower customers who want to ask for more secure software
- Provide a starting point for in-house software shops to measure their own progress



CWE List

[Full Dictionary View](#)
[Development View](#)
[Research View](#)
[Reports](#)

About

[Sources](#)
[Process](#)
[Documents](#)

Community

[Related Activities](#)
[Discussion List](#)
[Research](#)
[CWE/SANS Top 25](#)
[CWSS](#)

News

[Calendar](#)
[Free Newsletter](#)

Compatibility

[Program](#)
[Requirements](#)
[Declarations](#)
[Make a Declaration](#)

Contact Us

[Search the Site](#)

2010 CWE/SANS Top 25 Most Dangerous Software Errors

Copyright © 2010

The MITRE Corporation

<http://cwe.mitre.org/top25/>**Document version:** 1.06 ([pdf](#))**Date:** September 27, 2010**Project Coordinators:**

Bob Martin (MITRE)
 Mason Brown (SANS)
 Alan Paller (SANS)
 Dennis Kirby (SANS)

Document Editor:

Steve Christey (MITRE)

Section Contents**CWE/SANS Top 25**

[Contributors](#)
[Supporting Quotes](#)
[Monster Mitigations](#)
[Focus Profiles](#)
[On the Cusp](#)
[Documents & Podcasts](#)
[Training Materials](#)
[Top 25 FAQ](#)
[Top 25 Process](#)
[Change Log](#)

SANS News Release**Section Archives****2009 CWE/SANS Top 25**

[Supporting Quotes](#)
[Contributors](#)
[On The Cusp](#)
[Change Log](#)

Introduction

The 2010 CWE/SANS Top 25 Most Dangerous Software Errors is a list of the most widespread and critical programming errors that can lead to serious software vulnerabilities. They are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.

The Top 25 list is a tool for education and awareness to help programmers to prevent the kinds of vulnerabilities that plague the software industry, by identifying and avoiding all-too-common mistakes that occur before software is even shipped. Software customers can use the same list to help them to ask for more secure software. Researchers in software security can use the Top 25 to

Robert C. Seacord	CERT	Ryan Barnett	Breach
Pascal Meunier	CERIAS, Purdue University	Antonio Fontes	New Ac
Matt Bishop	University of California, Davis	Mark Giovanni II	Micron
Kenneth van Wyk			
Masato Terada			
Sean Barnum			
Mahesh Saptarshi			
Cassio Goldschmidt			
Adam Hahn			
Jeff Williams			
Carsten Eiram			
Josh Drake			
Chuck Willis			
Michael Howard			
Bruce Lowenthal			
Mark J. Cox			
Jacob West			
Djenana Campara			
James Walden			
Frank Kim			
Chris Eng	Veracode, Inc.		
Chris Wysopal	Veracode, Inc.		

CWE - Top 25 Credited Contributors

http://cwe.mitre.org/top25/contributors.html



CWE Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

2010

Home > CWE/SANS Top 25 > Credited Contributors

[CWE List](#) [Credited Contributors](#)

Search by ID: [Go](#)

























Veracode, Inc.

2009


 CWE is a [Software Assurance](#) strategic initiative sponsored by the [National Cyber Security Division](#) of the U.S. Department of Homeland Security. This Web site is hosted by [The MITRE Corporation](#). Copyright 2010, The MITRE Corporation. CWE and the CWE logo are trademarks of The MITRE Corporation. Contact cwe@mitre.org for more information.

[Privacy policy](#)
[Terms of use](#)
[Contact us](#)

Section Contents

- CWE/SANS Top 25**
- Contributors
- Supporting Quotes
- Monster Mitigations
- Focus Profiles
- On the Cusp
- Documents & Podcasts
- Training Materials
- Top 25 FAQ
- Top 25 Process
- Change Log
- SANS News Release**

Section Archives

- 2009 CWE/SANS Top 25
- Supporting Quotes
- Contributors
- On The Cusp
- Change Log

Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

- [CWE-20](#): Improper Input Validation
- [CWE-116](#): Improper Encoding or Escaping of Output
- [CWE-89](#): Failure to Preserve SQL Query Structure (aka 'SQL Injection')
- [CWE-79](#): Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
- [CWE-78](#): Failure to Preserve OS Command Structure (aka 'OS Command Injection')
- [CWE-319](#): Cleartext Transmission of Sensitive Information
- [CWE-352](#): Cross-Site Request Forgery (CSRF)
- [CWE-362](#): Race Condition
- [CWE-209](#): Error Message Information Leak

Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

- [CWE-119](#): Failure to Constrain Operations within the Bounds of a Memory Buffer
- [CWE-642](#): External Control of Critical State Data
- [CWE-73](#): External Control of File Name or Path
- [CWE-426](#): Untrusted Search Path
- [CWE-94](#): Failure to Control Generation of Code (aka 'Code Injection')
- [CWE-494](#): Download of Code Without Integrity Check
- [CWE-404](#): Improper Resource Shutdown or Release
- [CWE-665](#): Improper Initialization
- [CWE-682](#): Incorrect Calculation

Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

- [CWE-285](#): Improper Access Control (Authorization)
- [CWE-327](#): Use of a Broken or Risky Cryptographic Algorithm
- [CWE-259](#): Hard-Coded Password
- [CWE-732](#): Insecure Permission Assignment for Critical Resource
- [CWE-330](#): Use of Insufficiently Random Values
- [CWE-250](#): Execution with Unnecessary Privileges
- [CWE-602](#): Client-Side Enforcement of Server-Side Security

Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

For each weakness, its ranking in the general list is provided in square brackets.

Rank	CWE ID	Name
[1]	CWE-79	Failure to Preserve Web Page Structure ('Cross-site Scripting')
[2]	CWE-89	Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')
[4]	CWE-352	Cross-Site Request Forgery (CSRF)
[8]	CWE-434	Unrestricted Upload of File with Dangerous Type
[9]	CWE-78	Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')
[17]	CWE-209	Information Exposure Through an Error Message
[23]	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[25]	CWE-362	Race Condition

Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

Rank	CWE ID	Name
[3]	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[7]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[12]	CWE-805	Buffer Access with Incorrect Length Value
[13]	CWE-754	Improper Check for Unusual or Exceptional Conditions
[14]	CWE-98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
[15]	CWE-129	Improper Validation of Array Index
[16]	CWE-190	Integer Overflow or Wraparound
[18]	CWE-131	Incorrect Calculation of Buffer Size
[20]	CWE-494	Download of Code Without Integrity Check
[22]	CWE-770	Allocation of Resources Without Limits or Throttling

Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

Rank	CWE ID	Name
[5]	CWE-285	Improper Access Control (Authorization)
[6]	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[10]	CWE-311	Missing Encryption of Sensitive Data
[11]	CWE-798	Use of Hard-coded Credentials
[19]	CWE-306	Missing Authentication for Critical Function
[21]	CWE-732	Incorrect Permission Assignment for Critical Resource
[24]	CWE-327	Use of a Broken or Risky Cryptographic Algorithm



2 **CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')**

Summary

Weakness Prevalence	High	Consequences	Data loss, Security bypass
Remediation Cost	Low	Ease of Detection	Easy
Attack Frequency	Often	Attacker Awareness	High

Discussion

These days, it seems as if software is all about the data: getting it into the database, pulling it from the database, massaging it into information, and sending it elsewhere for fun and profit. If attackers can influence the SQL that you use to communicate with your database, then suddenly all your fun and profit belongs to them. If you use SQL queries in security controls such as authentication, attackers could alter the logic of those queries to bypass security. They could modify the queries to steal, corrupt, or otherwise change your underlying data. They'll even steal data one byte at a time if they have to, and they have the patience and know-how to do so.

[Technical Details](#) | [Code Examples](#) | [Detection Methods](#) | [References](#)

Prevention and Mitigations

Architecture and Design

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using persistence layers such as Hibernate or Enterprise Java Beans, which can provide significant protection against SQL injection if used properly.

Architecture and Design

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

Process SQL queries using prepared statements, parameterized queries, or stored procedures. These features should accept parameters or variables and support strong typing. Do not dynamically construct and execute query strings within these features using "exec" or similar functionality, since you may

Monster Mitigations

These mitigations will be effective in eliminating or reducing the severity of the Top 25. These mitigations will also address many weaknesses that are not even on the Top 25. If you adopt these mitigations, you are well on your way to making more secure software.

A [Monster Mitigation Matrix](#) is also available to show how these mitigations apply to weaknesses in the Top 25.

ID	Description
M1	Establish and maintain control over all of your inputs.
M2	Establish and maintain control over all of your outputs.
M3	Lock down your environment.
M4	Assume that external components can be subverted, and your code can be read by anyone.
M5	Use industry-accepted security features instead of inventing your own.
GP1	(general) Use libraries and frameworks that make it easier to avoid introducing weaknesses.
GP2	(general) Integrate security into the entire software development lifecycle.
GP3	(general) Use a broad mix of methods to comprehensively find and prevent weaknesses.
GP4	(general) Allow locked-down clients to interact with your software.

M1	M2	M3	M4	M5	CWE
High		DiD	Mod		CWE-22 : Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
Mod	High	DiD	Ltd		CWE-78 : Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')
Mod	High		Ltd		CWE-79 : Failure to Preserve Web Page Structure ('Cross-site Scripting')
Mod	High	DiD	Ltd		CWE-89 : Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')
Mod		DiD	Ltd		CWE-98 : Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
Mod		DiD	Ltd		CWE-120 : Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
High		DiD	Ltd		CWE-129 : Improper Validation of Array Index
Mod		DiD	Ltd		CWE-131 : Incorrect Calculation of Buffer Size
Mod		DiD	Ltd		CWE-190 : Integer Overflow or Wraparound
Ltd	High	DiD	Mod		CWE-209 : Information Exposure Through an Error Message
		DiD	Mod	Mod	CWE-285 : Improper Access Control (Authorization)
		Mod		Mod	CWE-306 : Missing Authentication for Critical Function
		DiD			CWE-311 : Missing Encryption of Sensitive Data
				High	CWE-327 : Use of a Broken or Risky Cryptographic Algorithm
			Ltd		CWE-352 : Cross-Site Request Forgery (CSRF)
		DiD			CWE-362 : Race Condition
Mod		DiD	Mod		CWE-434 : Unrestricted Upload of File with Dangerous Type
		DiD			CWE-494 : Download of Code Without Integrity Check
Mod	Mod		Ltd		CWE-601 : URL Redirection to Untrusted Site ('Open Redirect')
	Ltd	DiD		Mod	CWE-732 : Incorrect Permission Assignment for Critical Resource
Mod	Ltd	DiD			CWE-754 : Improper Check for Unusual or Exceptional Conditions
Ltd		DiD	Ltd		CWE-770 : Allocation of Resources Without Limits or Throttling
		DiD	High	Mod	CWE-798 : Use of Hard-coded Credentials
Mod		DiD	Ltd		CWE-805 : Buffer Access with Incorrect Length Value
Mod		DiD	Mod	Mod	CWE-807 : Reliance on Untrusted Inputs in a Security Decision

Focus Profiles

The prioritization of items in the general Top 25 list is just that - general. The rankings, and even the selection of which items should be included, can vary widely depending on context. Ideally, each organization can decide how to rank weaknesses based on its own criteria, instead of relying on a single general-purpose list.

A separate document provides several "focus profiles" with their own criteria for selection and ranking, which may be more useful than the general list.

Name	Description
<u>On the Cusp: Weaknesses that Did Not Make the 2010 Top 25</u>	From the original nominee list of 41 submitted CWE entries, the Top 25 was selected. This "On the Cusp" profile includes the remaining 16 weaknesses that did not make it into the final Top 25.
<u>Educational Emphasis</u>	This profile ranks weaknesses that are important from an educational perspective within a school or university context. It focuses on the CWE entries that graduating students should know, including historically important weaknesses.
<u>Weaknesses by Language</u>	This profile specifies which weaknesses appear in which programming languages. Notice that most weaknesses are actually language-independent, although they may be more prevalent in one language or another.
<u>Weaknesses Typically Fixed in Design or Implementation</u>	This profile lists weaknesses that are typically fixed in design or implementation.
<u>Automated vs. Manual Analysis</u>	This profile highlights which weaknesses can be detected using automated versus manual analysis. Currently, there is very little public, authoritative information about the efficacy of these methods and their utility. There are many competing opinions, even among experts. As a result, these ratings should only be treated as guidelines, not rules.
<u>Weaknesses by Language</u>	This profile specifies which weaknesses appear in which programming languages. Notice that most weaknesses are actually language-independent, although they may be more prevalent in one language or another.
<u>For Developers with Established Software Security Practices</u>	This profile is for developers who have already established security in their practice. It uses votes from the major developers who contributed to the Top 25.
<u>Ranked by Importance - for Software Customers</u>	This profile ranks weaknesses based primarily on their importance, as determined from the base voting data that was used to create the general list. Prevalence is included in the scores, but it has much less weighting than importance.
<u>Weaknesses by Technical Impact</u>	This profile lists weaknesses based on their technical impact, i.e., what an attacker can accomplish by exploiting each weakness.

Background Details to Check Out

cwe.mitre.org/top25

- Process description
- Changelog for each revision
- On the Cusp – weaknesses that almost made it
- Appendices
 - **Selection Criteria and Supporting Fields**
 - **Threat Model for the Skilled, Determined Attacker**

On the Cusp: Other Weaknesses to Consider

Table of Contents

1. [Introduction](#)
2. [Weaknesses that did not have sufficient prevalence or severity](#)
3. [Weaknesses covered by more general entries](#)

Introduction

The CWE/SANS Top 25 is really just a starting point for developers. Many weaknesses were considered for inclusion on the Top 25, but some did not make it to the final list. Some were not considered to be severe enough; others were not considered to be prevalent enough. Sometimes, the Top 25 reviewers themselves had mixed opinions on whether a weakness should be added to the list or not.

With respect to severity, some Top 25 users may have a significantly different threat model. For example, software uptime may be critical to consumers who operate in critical infrastructure or e-commerce environments. However, in the threat model being used by the Top 25, availability is regarded as slightly less important than integrity and confidentiality.

With respect to prevalence, some Top 25 items may not be applicable to the class of software being developed. For example, cross-site scripting is specific to the Web, although analogs exist in other technologies. In other cases, developers may have already eliminated much of the Top 25 in past efforts, so they want to look for other weaknesses that may still be present in their software.

Some on-the-cusp items were omitted because they are already indirectly covered on the Top 25, usually by a more general entry. However, these would be important to consider as individual items.

For these reasons, users of the Top 25 should seriously consider including these weaknesses in their analyses.

[BACK TO TOP](#)

Weaknesses that did not have sufficient prevalence or severity

[26]	136	CWE-749 : Exposed Dangerous Method or Function
		<i>Just 2 points from the Top 25, possibly on the rise.</i>
[27]	129	CWE-307 : Improper Restriction of Excessive Authentication Attempts
		<i>Possibly squeezed off the Top 25 by cousins such as missing authentication.</i>
[28]	125	CWE-212 : Improper Cross-boundary Removal of Sensitive Data
		<i>Important when privacy is a main concern.</i>
[29]	124	CWE-330 : Use of Insufficiently Random Values
		<i>Not always security-relevant, but still dangerous if it is.</i>
[30]	120	CWE-59 : Improper Link Resolution Before File Access ('Link Following')
		<i>A burst in CVE statistics in 2008 shows that these can still be prevalent if focused attention is paid to them.</i>
[31] (tie)	120	CWE-134 : Uncontrolled Format String
		<i>Usually easily findable, and code execution possibilities have been reduced due to compiler changes, e.g. removal of support for "%n" sequences.</i>
[32]	119	CWE-476 : NULL Pointer Dereference
		<i>Typically cause a denial of service in C/C++ but, for certain Linux kernels and possibly other environments, exploitable for code execution.</i>
[33] (tie)	119	CWE-681 : Incorrect Conversion between Numeric Types
		<i>May be on the rise in future years, especially in transitions from 32-bit to 64-bit architectures.</i>
[34]	118	CWE-426 : Untrusted Search Path
		<i>Prevalence is uncertain.</i>
[35]	116	CWE-454 : External Initialization of Trusted Variables or Data Stores
		<i>High prevalence in PHP environments with register_globals enabled, or by programmers who are not familiar with the effectiveness of reverse engineering, or the many ways that inputs can be modified.</i>
[36]	114	CWE-416 : Use After Free
		<i>Likely on the rise in future years.</i>
[37] (tie)	114	CWE-772 : Missing Release of Resource after Effective Lifetime
		<i>Important when prevention of denial of service is critical.</i>
[38]	106	CWE-799 : Improper Control of Interaction Frequency
		<i>Important when prevention of denial of service is critical. Also a critical component of brute force attacks against security features.</i>
[39]	100	CWE-456 : Missing Initialization
		<i>Not always security-relevant; also, easily findable and fixable with modern compilers and code scanners.</i>
[40]	91	CWE-672 : Operation on a Resource after Expiration or Release
		<i>Sometimes catchable by the compiler, but may increase in future years.</i>
[41]	77	CWE-804 : Guessable CAPTCHA
		<i>Not very prevalent since the use of CAPTCHA is not very prevalent, and importance is generally less than that of other security features such as encryption and authentication.</i>

Frequently Asked Questions (FAQ)

How is this different from the OWASP Top Ten?

The short answer is that the OWASP Top Ten covers more general concepts and is focused on web applications. The CWE Top 25 covers a broader range of issues than what arise from the web-centric view of the OWASP Top Ten, such as buffer overflows. Also, one goal of the CWE Top 25 is to be at a level that is directly actionable to programmers, so it contains more detailed issues than the categories being used in the Top Ten. There is some overlap, however, since web applications are so prevalent, and some issues in the Top Ten have general applications to all classes of software.

How are the weaknesses prioritized on the list?

With the exception of Input Validation being listed as number 1 (partially for educational purposes), there is no concrete prioritization. Prioritization differs widely depending on the audience (e.g. web application developers versus OS developers) and the risk tolerance (whether code execution, data theft, or denial of service are more important). It was also believed that the use of categories would help the organization of the document, and prioritization would impose a different ordering.

Why are you including overlapping concepts like input validation and XSS, or incorrect calculation and buffer overflows? Why do you have mixed levels of abstraction?

While it would have been ideal to have a fixed level of abstraction and no overlap between weaknesses, there are several reasons why this was not achieved.

Contributors sometimes suggested different CWE identifiers that were closely related. In some cases, this difference was addressed by using a more abstract CWE identifier that covered the relevant cases.

In other situations, there was strong advocacy for including lower-level issues such as SQL injection and cross-site scripting, so these were added. The general trend, however, was to use more abstract weakness types.

While it might be desired to minimize overlap in the Top 25, many vulnerabilities actually deal with the interaction of 2 or more weaknesses. For example, external control of user state data (CWE-642) could be an important weakness that enables cross-site scripting (CWE-79) and SQL injection (CWE-89). To eliminate overlap in the Top 25 would lose some of this important subtlety.

Finally, it was a conscious decision that if there was enough prevalence and severity, design-related weaknesses would be included. These are often thought of as being more abstract than weaknesses that arise during implementation.

The Top 25 list tries to strike a delicate balance between usability and relevance, and we believe that it does so, even with this apparent imperfection.

Why don't you use hard statistics to back up your claims?

The appropriate statistics simply aren't publicly available. The publicly available statistics are either too high-level or not comprehensive enough. And none of them are comprehensive across all software types and environments.

People are Starved for Simplicity

Google Analytics

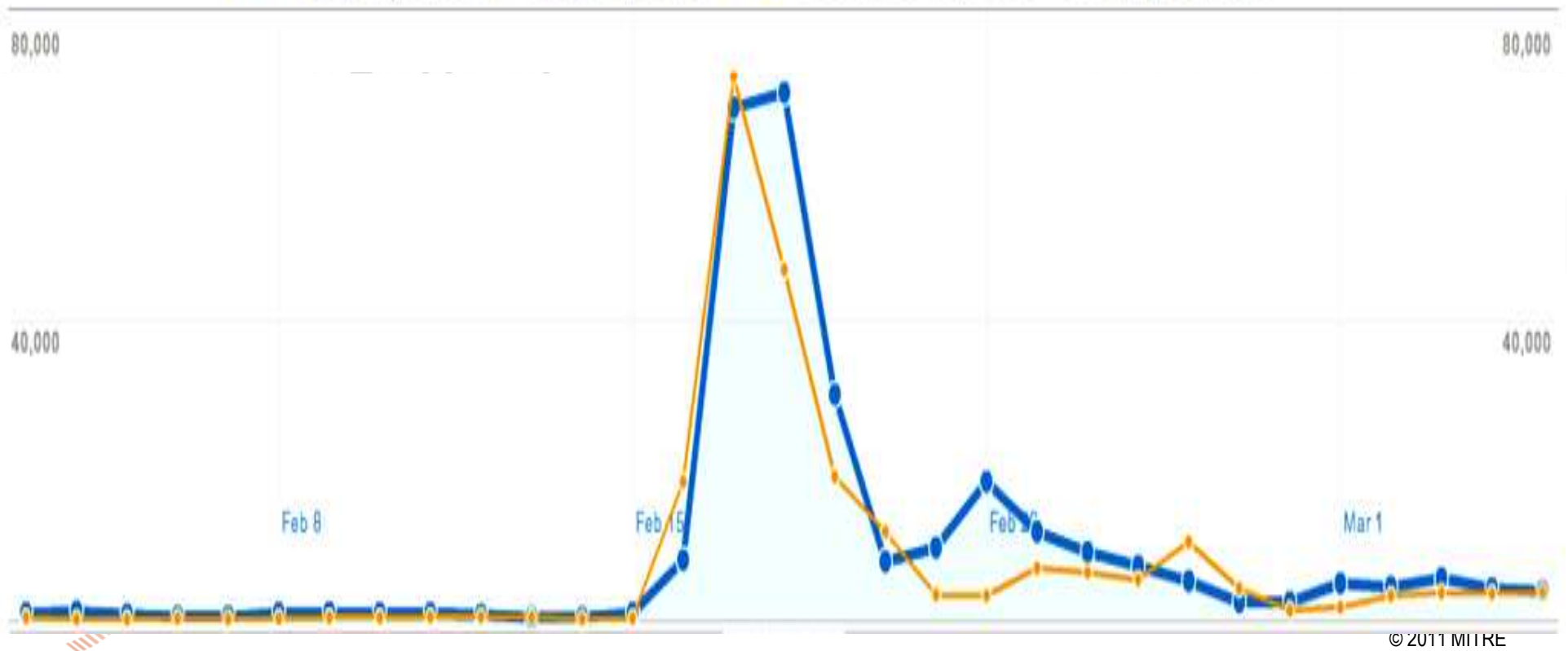
ramartin@mitre.org | [Settings](#) | [My Account](#) | [Help](#) | [Sign Out](#)

[Analytics Settings](#)

[View Reports:](#)

[My Analytics Accounts:](#)

■● February 3, 2010 - March 5, 2010 ■● December 30, 2008 - January 29, 2009



Top 25 Series – Summary and Links

Posted by Frank Kim on April 6, 2010 – 3:41 pm
Filed under Top25

As requested here are the links to all the posts on the Top 25 Most Dangerous Programming Errors. Please let us know if you have any suggestions or comments.

- 1 – [Cross-Site Scripting \(XSS\)](#)
- 2 – [SQL Injection](#)
- 3 – [Classic Buffer Overflow](#)
- 4 – [Cross-Site Request Forgery \(CSRF\)](#)
- 5 – [Improper Access Control \(Authorization\)](#)
- 6 – [Reliance on Untrusted Inputs in a Security Decision](#)
- 7 – [Path Traversal](#)
- 8 – [Unrestricted Upload of Dangerous File Type](#)
- 9 – [OS Command Injection](#)
- 10 – [Missing Encryption of Sensitive Data](#)
- 11 – [Hardcoded Credentials](#)
- 12 – [Buffer Access with Incorrect Length Value](#)
- 13 – [PHP File Inclusion](#)
- 14 – [Improper Validation of Array Index](#)
- 15 – [Improper Check for Unusual or Exceptional Conditions](#)
- 16 – [Information Exposure Through an Error Message](#)
- 17 – [Integer Overflow Or Wraparound](#)
- 18 – [Incorrect Calculation of Buffer Size](#)
- 19 – [Missing Authentication for Critical Function](#)
- 20 – [Download of Code Without Integrity Check](#)
- 21 – [Incorrect Permission Assignment for Critical Response](#)
- 22 – [Allocation of Resources Without Limits or Throttling](#)
- 23 – [Open Redirect](#)
- 24 – [Use of a Broken or Risky Cryptographic Algorithm](#)
- 25 – [Race Conditions](#)

Pat on Some Thoughts About Passwords

Jim on Seven Security (Mis)Configurations in Java web.xml Files

Nick Owen on Some Thoughts About Passwords

ARCHIVES

Select Month

META

Log in

Entries RSS

Comments RSS

WordPress.org





The Security Development Lifecycle

Recent Posts

- [SDL Threat Modeling Tool 3.1.4 ships!](#)
- [Early Days of the SDL, Part Four](#)
- [Early Days of the SDL, Part Three](#)
- [Early Days of the SDL, Part Two](#)
- [Early Days of the SDL, Part One](#)

Tags

- [Common Criteria](#) [Crawl Walk Run](#) [Privacy](#) [SDL](#) [SDL Pro](#)
- [Network](#) [Security Assurance](#)
- [Security Blackhat](#) [SDL](#) [threat modeling](#)

News

About Us

- [Adam Shostack](#)
- [Bryan Sullivan](#)
- [David Ladd](#)
- [Jeremy Dallman](#)
- [Michael Howard](#)
- [Steve Lipner](#)

Blogroll

- [BlueHat Security Briefings](#)

SDL and the CWE/SANS Top 25

Bryan here. The security community has been buzzing since SANS and MITRE's joint announcement earlier this month of their list of the [Top 25 Most Dangerous Programming Errors](#). Now, I don't want to get into a debate in this blog about whether this new list will become the new de facto standard for analyzing security vulnerabilities (or indeed, whether it already has become the new standard). Instead, I'd like to present an overview of how the Microsoft SDL maps to the CWE/SANS list, just May.

Michael and I have written coverage of the Top 25 and believe that the results of the 25 were developed independently root them out of the software analysis white paper and guidance around every made many of the same for you to download and

Below is a summary of how see the SDL covers every them (race conditions and by multiple SDL requirements tools to prevent or detect

CWE	Title
20	Improper Input Validation
116	Improper Encoding or Escaping of Output

CWE	Title	Education?	Manual Process?	Tools?	Threat Model?
20	Improper Input Validation	Y	Y	Y	Y
116	Improper Encoding or Escaping of Output	Y	Y	Y	
89	Failure to Preserve SQL Query Structure (aka SQL Injection)	Y	Y	Y	
79	Failure to Preserve Web Page Structure (aka Cross-Site Scripting)	Y	Y	Y	
78	Failure to Preserve OS Command Structure (aka OS Command Injection)	Y		Y	
319	Cleartext Transmission of Sensitive Information	Y			Y
352	Cross-site Request Forgery (aka CSRF)	Y		Y	
362	Race Condition	Y			
209	Error Message Information Leak	Y	Y	Y	
119	Failure to Constrain Memory Operations within the Bounds of a Memory Buffer	Y	Y	Y	
642	External Control of Critical State Data	Y			Y
73	External Control of File Name or Path	Y	Y	Y	
426	Untrusted Search Path	Y		Y	
94	Failure to Control Generation of Code (aka 'Code Injection')	Y	Y		
494	Download of Code Without Integrity Check				Y
404	Improper Resource Shutdown or Release	Y		Y	
665	Improper Initialization	Y		Y	
682	Incorrect Calculation	Y		Y	
285	Improper Access Control (Authorization)	Y	Y		Y
327	Use of a Broken or Risky Cryptographic Algorithm	Y	Y	Y	
259	Hard-Coded Password	Y	Y	Y	Y
732	Insecure Permission Assignment for Critical Resource	Y	Y		
330	Use of Insufficiently Random Values	Y	Y	Y	
250	Execution with Unnecessary Privileges	Y	Y		Y
602	Client-Side Enforcement of Server-Side Security	Y			Y

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 several times here, but review all missions and ACLs on all objects you create in the file system configuration stores such as Windows Vista and later, and change any default ACL in the system or registry unless you intend to weaken the ACL.

CWE-330: Use of Insufficiently Random Values

Identify all the random number generators in your code and determine which, if any, generate passwords, or some other secret. Make sure the code generating random numbers is cryptographically random and not a deterministic pseudorandom generator like the C runtime `rand()` function. Using functions like `rand()` is fine, 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 (on Linux, Unix, Mac OS X) or system (Windows) ask yourself, "Why?" Sometimes the answer is totally valid because the code must perform a privileged operation, but sometimes 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, opening a port below 1024 in a Linux application requires the code be run as root, but after that,

Basic Training

portant that the file and path do not exist before you access a file or path. Restrict what content or filename. As a view, look for a file or accesses file and make sure the name is appropriate to valid data. Remember "known good" is a better way to a

CWE-426: Untrusted

Old versions searched the root directory filenames, which problems if the had a weak password, weak or aren't committed no guarantee won't use searches or a permission source, environment variables is a remedy is to use path, but this is internationalization items—for example Vista, the API doesn't exist in version of Windows named or programming system correct path.

CWE-94: Failure to Generation

It's common to see code injection vulnerabilities in JavaScript code that builds a string dynamically and passes it to `eval()` to execute. 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.

(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 Sunny worm). Also, research into Web-related vulnerabilities has progressed substantially over the past few years, with new ways to attack systems regularly uncovered. For pure 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 obviously be protected at rest and while on the wire. The best solution to this vulnerability is to use a well-tested technology such as SSL/TLS or IPsec. Don't (ever!) create your own communication method and cryptographic defense. 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.

time. Fuzz testing is also effective at detecting CWE-665.

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 accounts. Correct use of synchronization primitives (mutexes, semaphores, critical sections) is similarly important.

CWE-642: External Control of Critical State

Unprotected state information such as profile data or configuration, is subject to attack. It's important to protect it by using the appropriate control lists (ACLs) or permissions for persistent data and sets of cryptographic defenses, a hashed message authentication code (HMAC), for one-time data. You can use an HMAC for persistent data as well.

CWE-73: External Control of Filename or Path

Attackers might be able to specify arbitrary file data if they the data that's used as a path or path name. It's critical

the very least, look for terms like "pwd" and "password" and make sure you have no hard-coded passwords or secret data in the code. You should also store this data in a secure location within the operating system. By secure, I mean protect it with an appropriate permission or encrypt it and protect the encryption key with an appropriate permission.

CWE-119: Failure to Constrain Memory Operations

The dreaded buffer overflow is a scourge of C and C++ and every vulnerability type has more headaches than buffer runs. The best way to solve the 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 because they don't offer direct access to memory. For C and C++ applications, developers should use "known bad" functions such as `strcpy` (for example, `strcpy_s`, `strcat_s`, `strcat`, `strcpy`, `strcat`, `strcpy_s`, `strcat_s`, and `strcpy_s`) and secure versions. Visual C++ has many weak APIs at compile and you should strive to use the more complex. Also, fuzz testing and static analysis can help identify potential buffer overruns, operating-system-level issues such as address space layout randomization and no execute can help reduce the chance that a buffer overrun is exploited.

CWE-642: External Control of Critical State

Unprotected state information such as profile data or configuration, is subject to attack. It's important to protect it by using the appropriate control lists (ACLs) or permissions for persistent data and sets of cryptographic defenses, a hashed message authentication code (HMAC), for one-time data. You can use an HMAC for persistent data as well.

CWE-73: External Control of Filename or Path

Attackers might be able to specify arbitrary file data if they the data that's used as a path or path name. It's critical

Basic Training

Editors: Richard Ford, rford@se.it.edu
Michael Howard, mikehow@microsoft.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

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

MICHAEL HOWARD
Microsoft

from industry, government, and academia who provided input to the document.

CWE, which stands for Common Weakness Enumeration, is a project sponsored by the National Cyber Security Division of the US Department of Homeland Security to classify security bugs. It assigns a unique number to weakness types such as buffer overruns or cross-site scripting bugs (for example, CWE-327 is "Use of a Broken or Risky Cryptographic Algorithm"). Shortly after the Top 25 list's release, Microsoft unveiled a document entitled, "The Microsoft SDL and the CWE/SANS Top 25," to explain how Microsoft's security processes can help prevent the worst offenders (<http://blogs.msdn.com/sdl/archive/2009/01/27/sdl-and-the-cwe-sans-top-25.aspx>).

Full disclosure: I'm one of that document's coauthors, but my purpose here isn't to regurgitate the Microsoft piece. Rather, my goal is to describe some best practices that can help you eliminate the CWE Top 25 vulnerabilities in your own development environment and products. It's also important to understand that addressing the weak-

nesses in the list doesn't imply your software is secure from all forms of attack; there are plenty more vulnerability types to worry about!

CWE-20: Improper Input Validation

The vast majority of serious security vulnerabilities are input validation issues: buffer overruns, SQL injection, and cross-site scripting bugs come immediately to mind. Developers simply trust the incoming data instead of understanding that they must analyze the input for validity. I can't stress this enough—if developers simply learned to never trust incoming data (in terms of format, content and size), many serious bugs would go away. The core lesson here is for developers to carefully validate input and for designers to understand how they can build their systems to protect input such that only trusted users can manipulate the data.

CWE-116: Improper Output Encoding

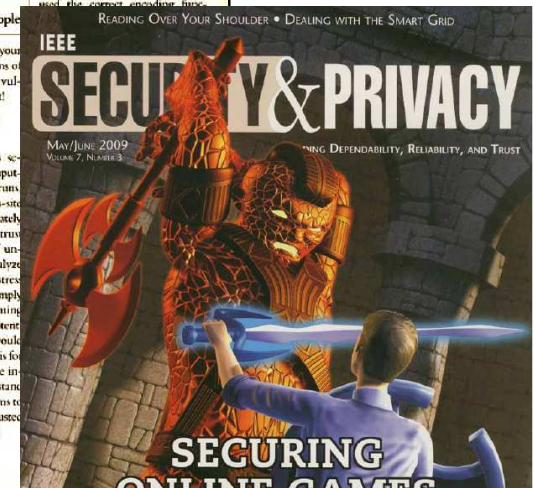
You could really isn't

Basic Training

68

Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

MICHAEL HOWARD



The Top 25 is not...

- A silver bullet
- A guarantee of software health
- A perfect match for your unique needs
- As simple as it seems
- The only thing to include in contract language
- Completely found by tools

The Top 25 is...

- A mechanism for awareness
- A trigger of questions
- A place for mitigations
- A conversation starter
- A first step on the long road to software assurance

CWE Top 25 2011

- Starting this week
- Utilizing the Common Weakness Scoring System (CWSS 0.2) as under-pinning
- Will have numerous “Top 25’s”
 - **Including one for Web Applications**
- Final "master" Top 25 list, will leverage combined score from multiple vignettes.
- No fixed date for release of the 2011 Top 25 at this point, may take 2 to 3 months.



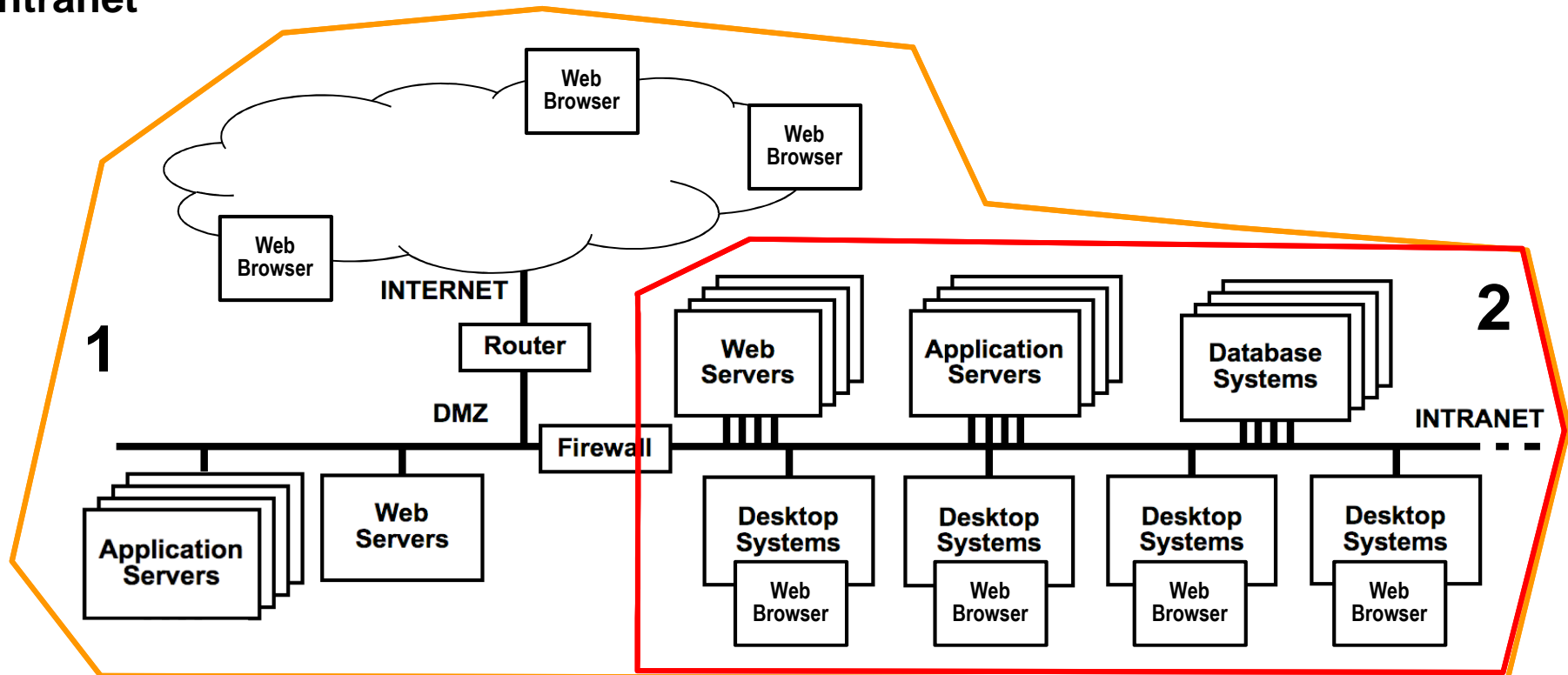
Common Weakness Scoring System (CWSS)

Archetypes:

- Web Browser User Interface
- Web Servers
- Application Servers
- Database Systems
- Desktop Systems
- SSL
- Internet
- DMZ
- Intranet

Vignettes:

1. Web-based Retail Provider
2. Intranet resident health records management system of hospital





Questions?

ramartin@mitre.org