



Knowing Your Weaknesses: The (CWE) Initiative

Bob Martin

October 1, 2010

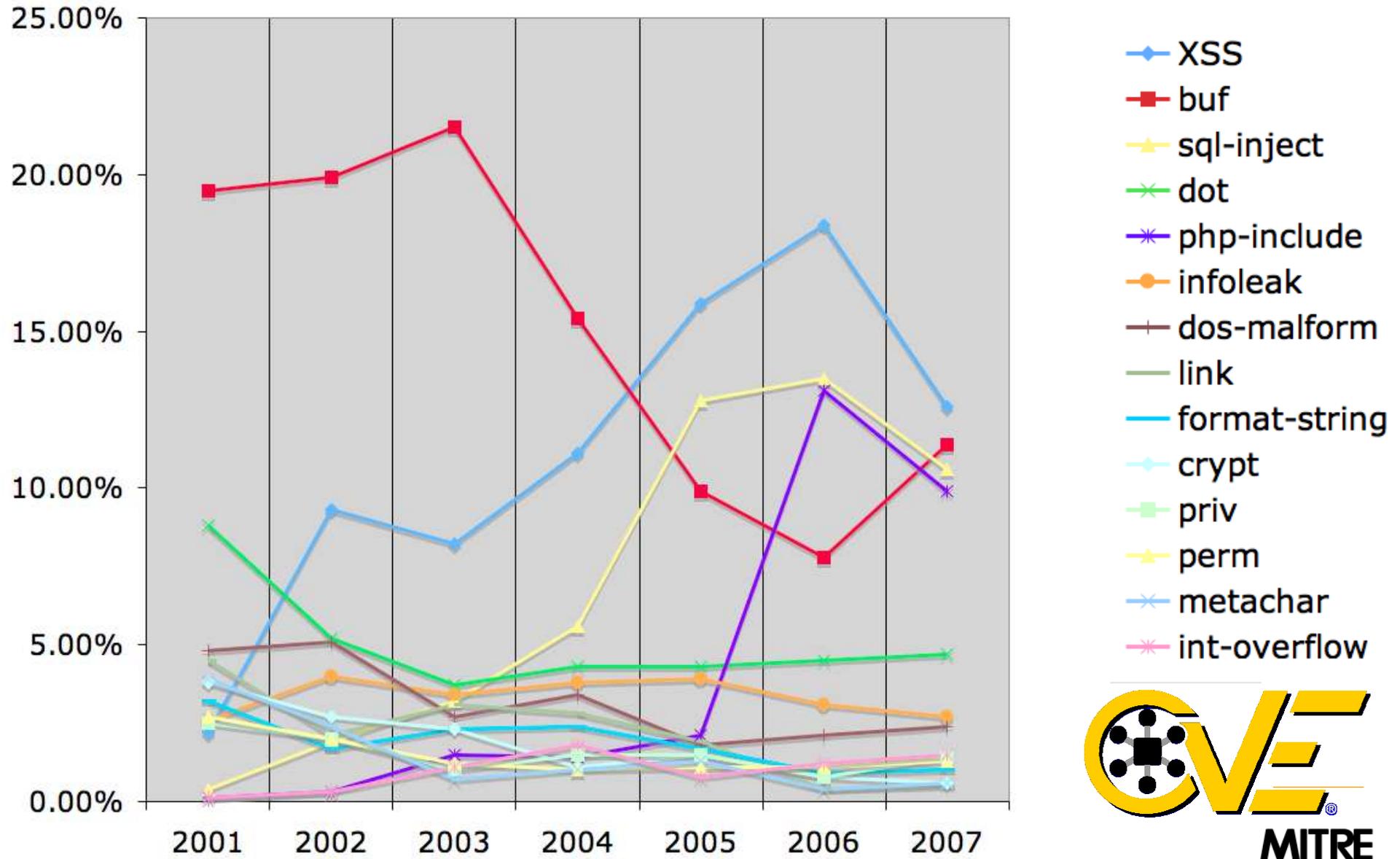


**If the weaknesses
in software were as
easy to spot and
their impact as
obvious as...**

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Using Unpublished Web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



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



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

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

Failure to Sanitize Directives in a Web Page (aka 'Cross-site scripting' (XSS)) (79)

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

Failure to Constrain Operations within the Bounds of an Allocated Memory Buffer (119)

- Unbounded Transfer ('Classic Buffer Overflow') (120)
- Write-what-where Condition (123)
- Boundary Beginning Violation ('Buffer Underwrite') (124)
- Out-of-bounds Read (125)
- Wrap-around Error (128)
- Unchecked Array Indexing (129)
- Incorrect Calculation of Buffer Size (131)
- Miscalculated Null Termination (132)
- Return of Pointer Value Outside of Expected Range (466)

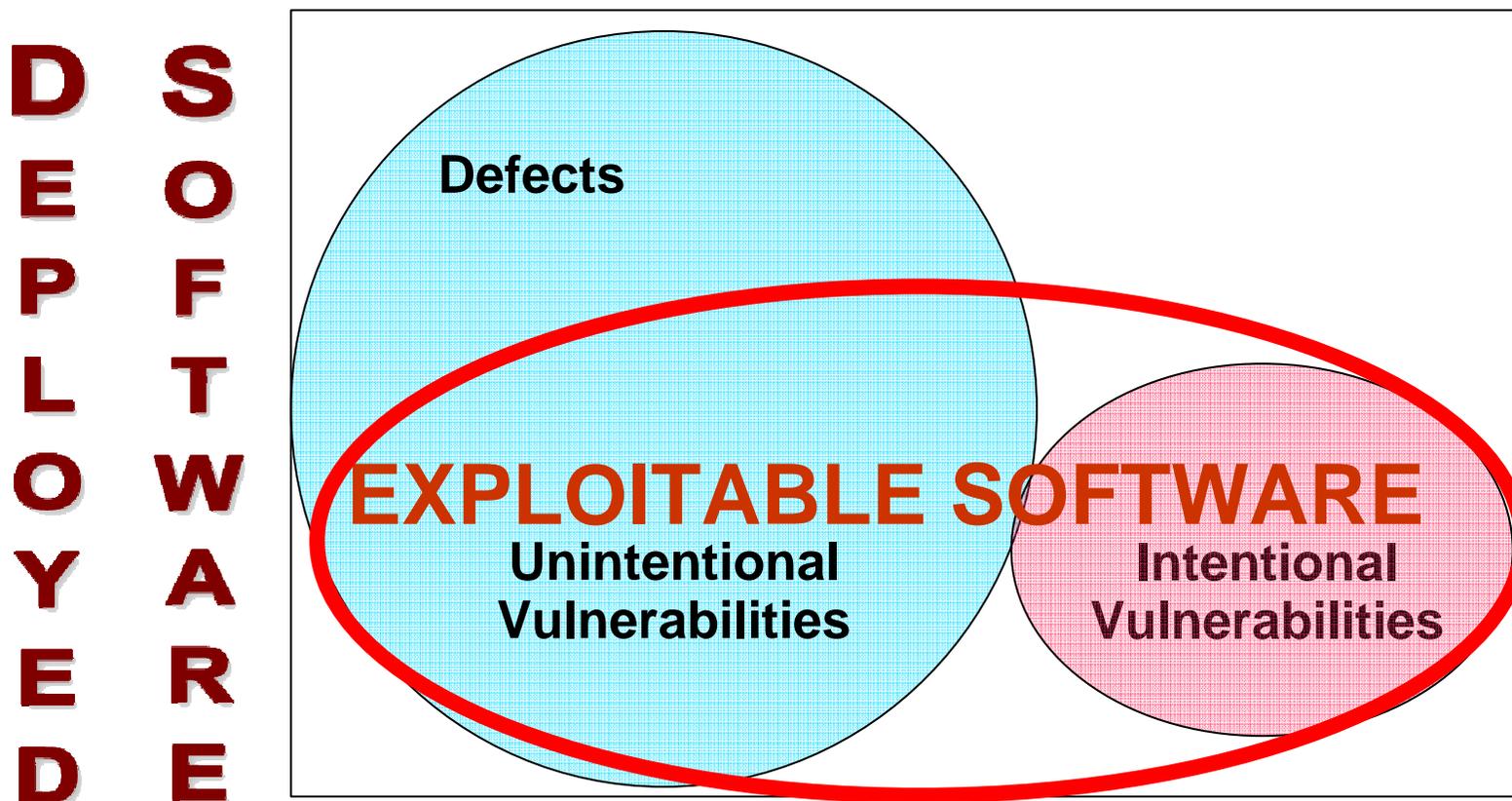
Path Traversal (22)

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

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)

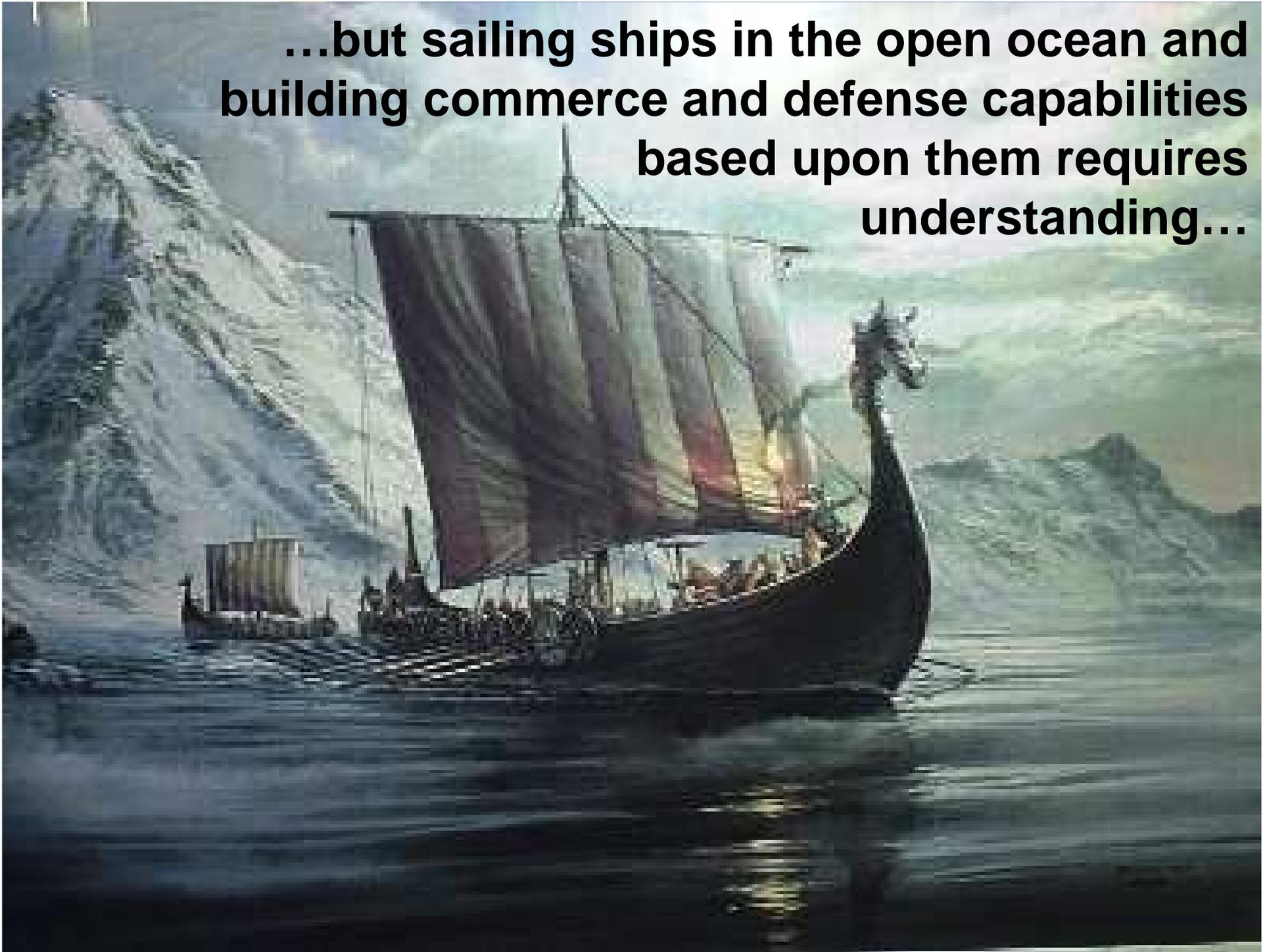
- dictionary of weaknesses
 - weaknesses that can lead to exploitable vulnerabilities (i.e. CVEs)
 - the things we don't want in our code, design, or architecture
 - web site with XML of content, sources of content, and process used
- structured views
 - currently provide hierarchical view into CWE dictionary content
 - will evolve to support alternate 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**

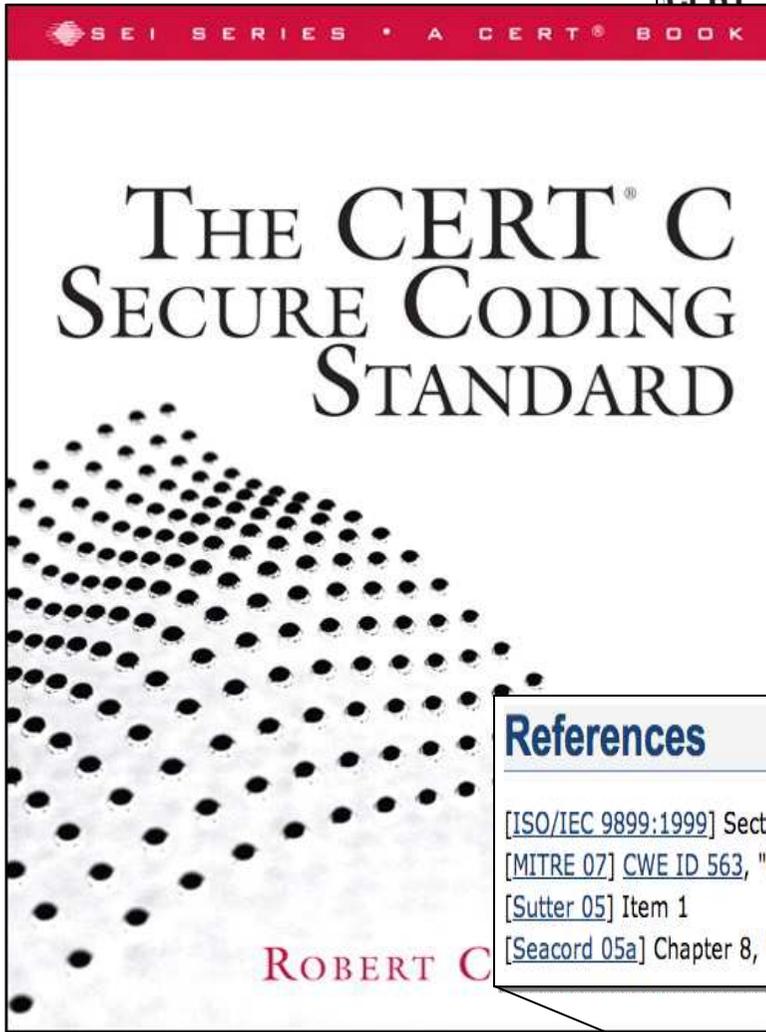
Building **Software**
only require a few
skills and basic
understanding...



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







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

Software Assurance Secure Systems Organizational Security Coordinated Response Training

ing Practices > ... > 49. Miscellaneous (MSC) > MSC00-CPP. Compile cleanly at high warning levels

Log In Sign Up

C++ Secure Coding Practices

MSC00-CPP. Compile cleanly at high warning levels

Added by [Justin Pincar](#), last edited by [Justin Pincar](#) on Oct 08, 2008 (view change) SHOW COMMENT
Labels: [unenforceable](#) [incomplete-cpp](#)

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

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

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

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

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

Exceptions

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

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

Risk Assessment

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

References

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

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>"



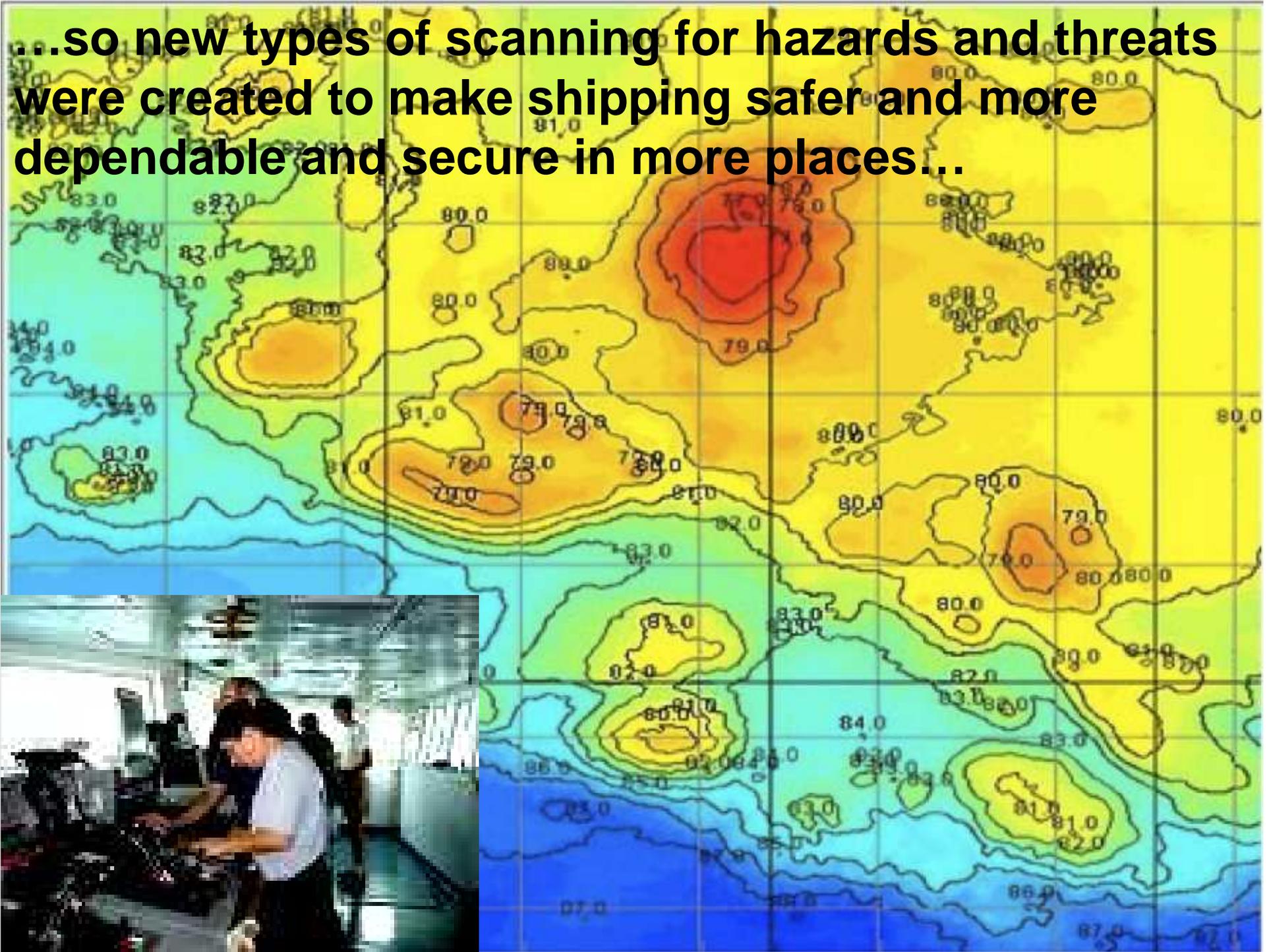
Homeland Security



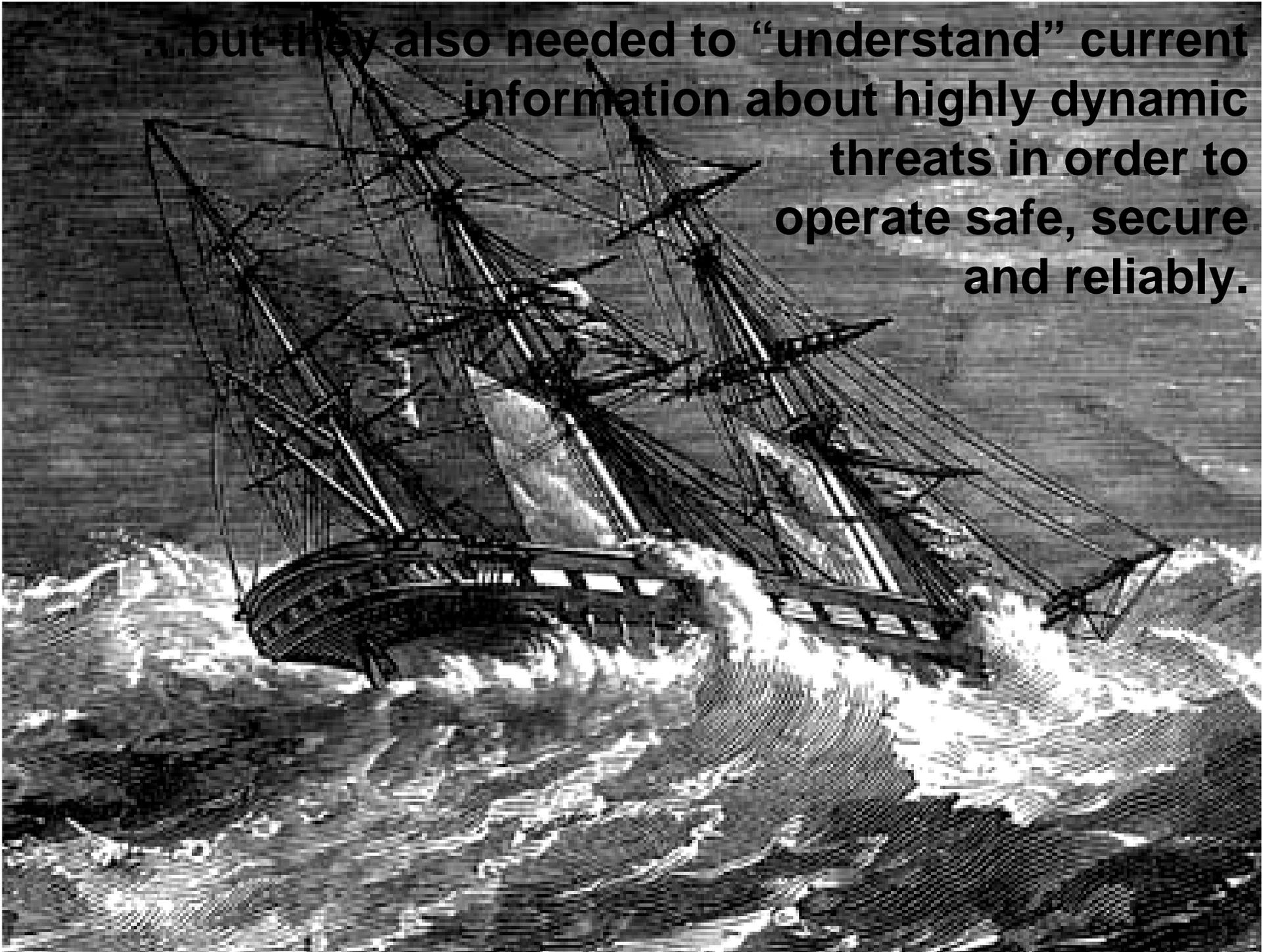
...some threats and hazards are unpredictable and dynamic...



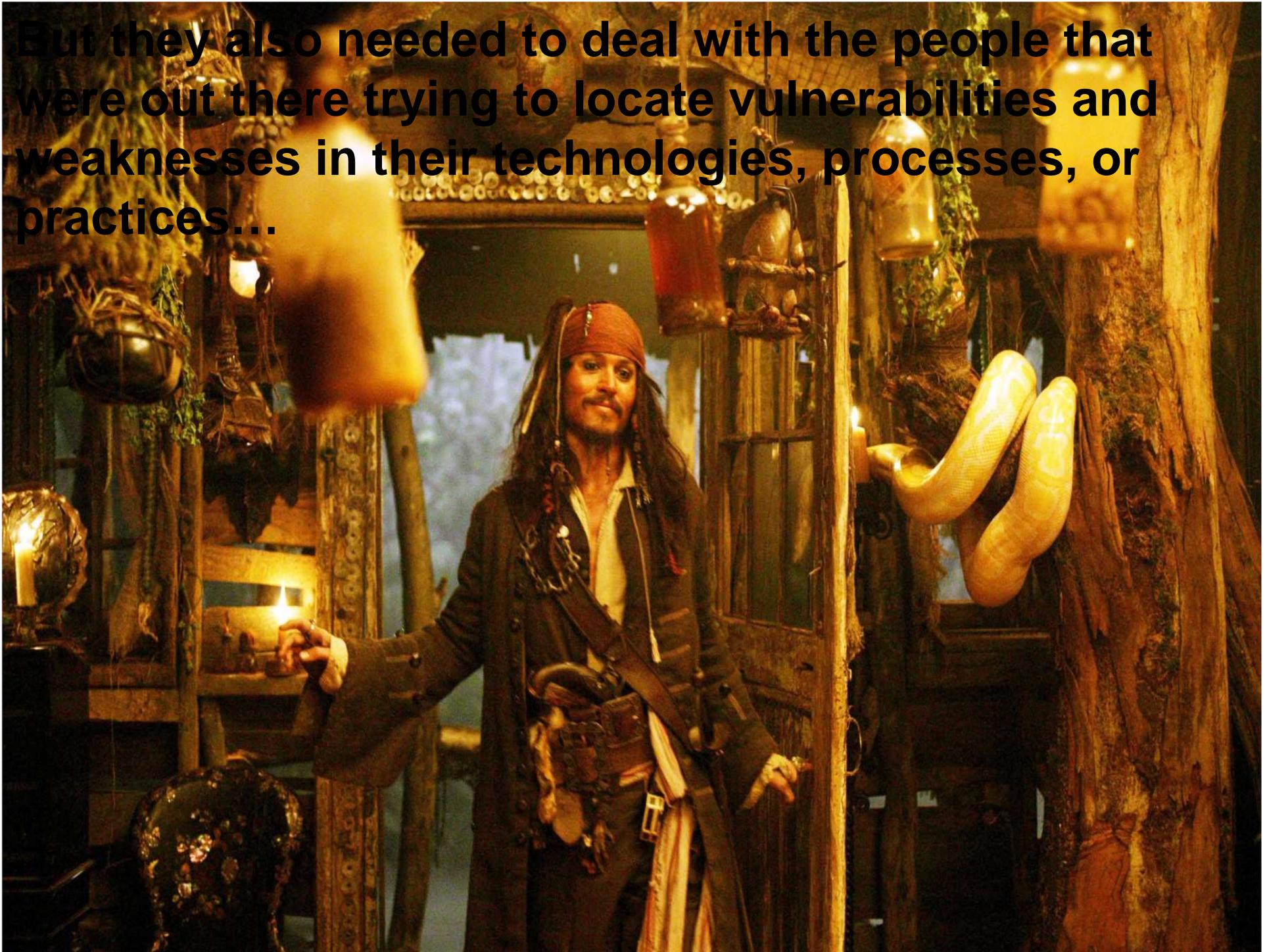
...so new types of scanning for hazards and threats were created to make shipping safer and more dependable and secure in more places...

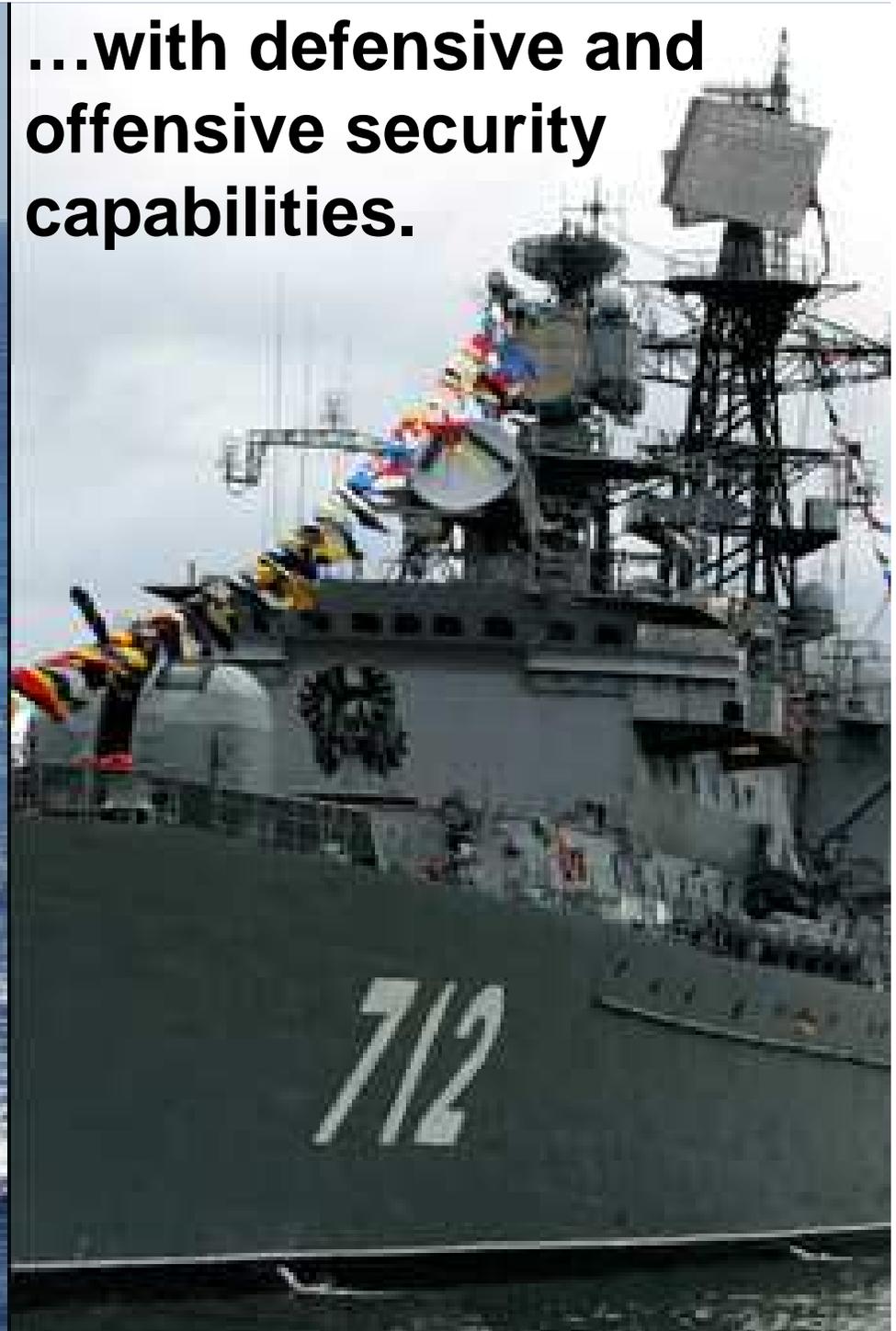


...but they also needed to “understand” current information about highly dynamic threats in order to operate safe, secure and reliably.

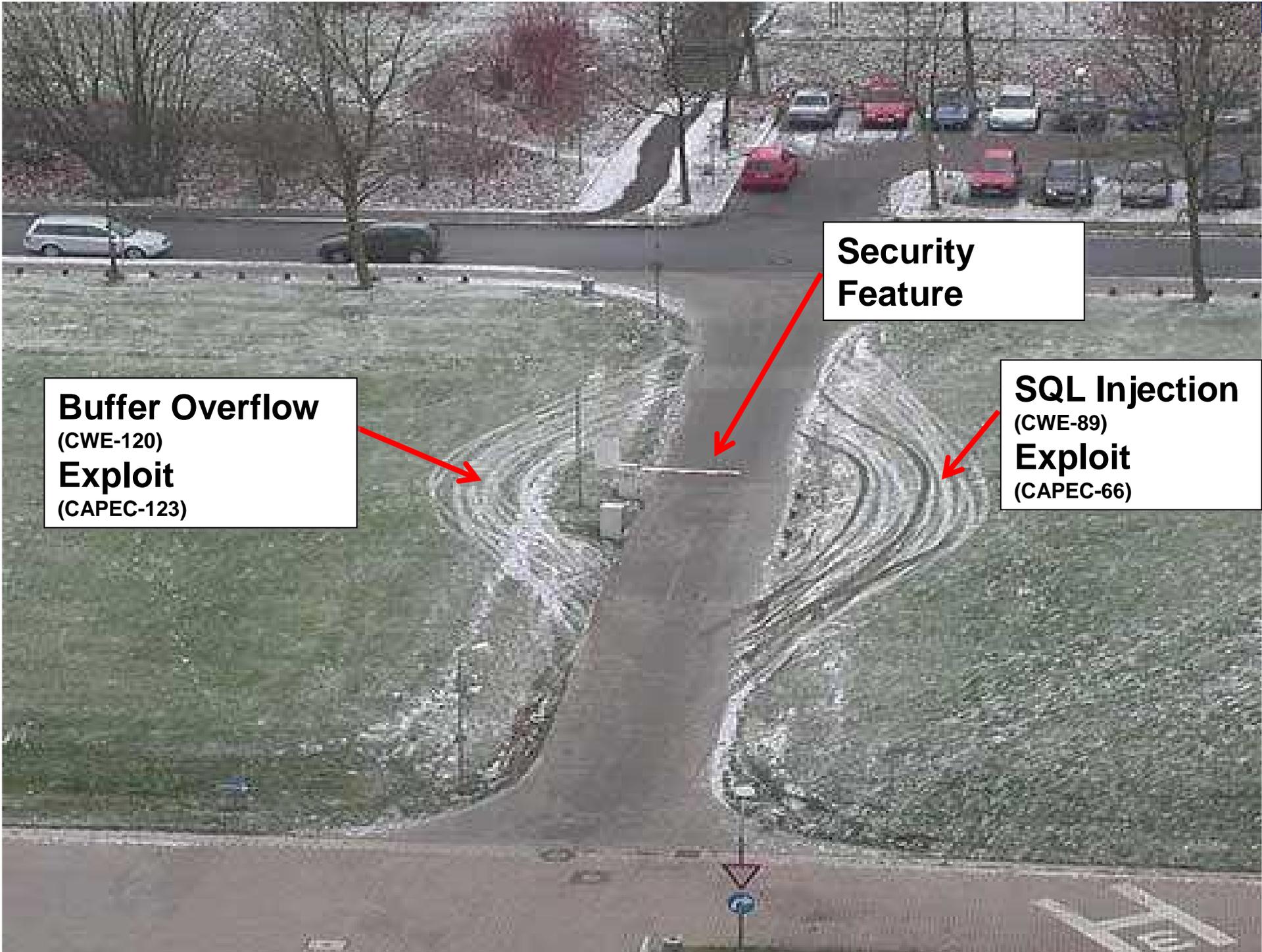


But they also needed to deal with the people that were out there trying to locate vulnerabilities and weaknesses in their technologies, processes, or practices...





...with defensive and offensive security capabilities.



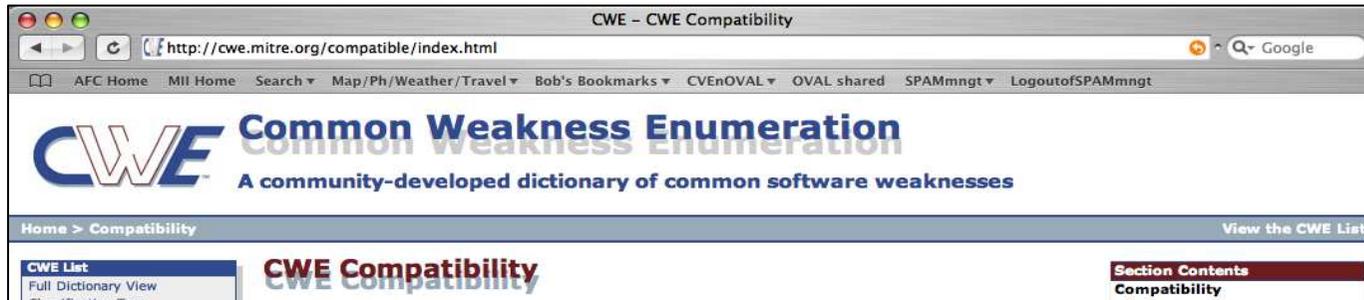
Buffer Overflow
(CWE-120)
Exploit
(CAPEC-123)

Security
Feature

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

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



● 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

OWASP Top Ten 2007 & 2010 use CWE refs

OWASP TOP 10



OWASP

The Open Web Application Security Project

OWASP Top 10 - 2010

The Ten Most Critical Web Application Security Risks

THE TEN MOST CRITICAL
WEB APPLICATION SECURITY RISKS

2007 UPDATE

© 2002-2007 OWASP Foundation
This document is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike license

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

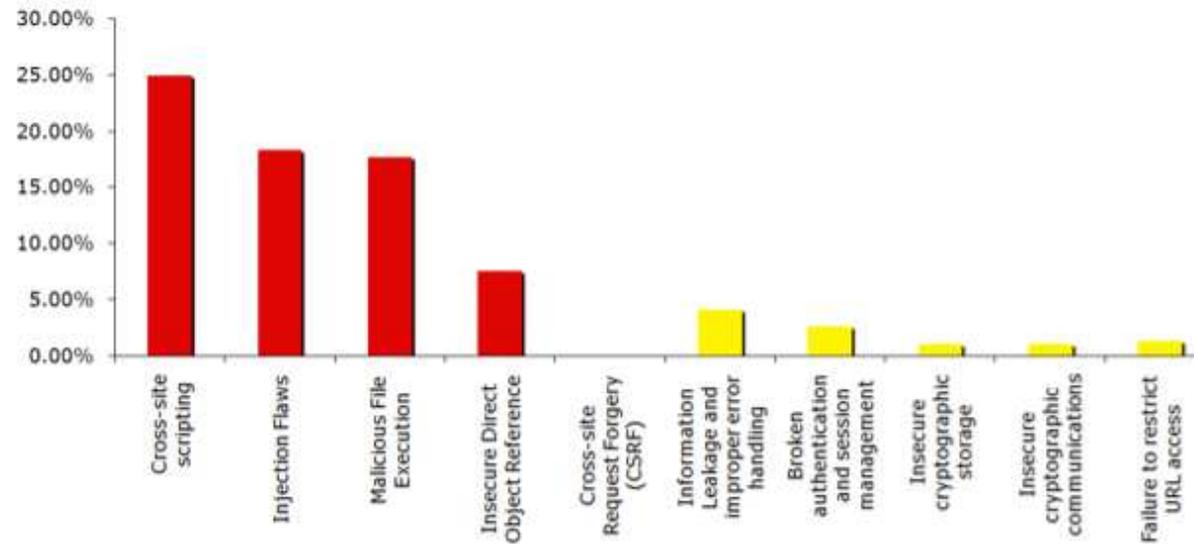


Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006

1
lease



reative Commons (CC) Attribution Share-Alike
Free version at <http://www.owasp.org>

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)

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.
Design: 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.

SAMATE Reference Dataset

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

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

» sign in register | Search... GO



SAMATE
Software Assurance Metrics and Tool Evaluation

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

Welcome to the NIST SAMATE Reference Dataset Project

The purpose of the SAMATE Reference Dataset (SRD) is to provide users, researchers, and developers with a set of known security flaws. This will allow end users to evaluate tools and tool designs, source code, binaries, etc., i.e. from all the phases of the software life cycle (written to test or generated), and "academic" (from students) test cases. This dataset includes known bugs and vulnerabilities. The dataset intends to encompass a wide variety of test cases, including known bugs and vulnerabilities. The dataset is anticipated to become a large-scale effort, gathering test cases about the SRD, including goals, structure, test suite selection, etc.

Browse, download, and search the SRD

Anyone can browse or search test cases and download selected cases. Please [click here](#) to view selected or all test cases. To find specific test cases, please [click here](#).

How to submit test cases

NIST Draft Special Publication 500-268

Source Code Security Analysis Tool Functional Specification Version 1.0

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

29 January, 2007

Michael Kass
Michael Koo

NIST Special Publications:

SP800-36	CVE
SP800-40	CVE, OVAL
SP800-42	CVE
SP800-44	CVE
SP800-51	CVE
SP800-53a	CVE, OVAL, CWE
SP800-61	CVE, OVAL
SP800-70	CVE, OVAL, CCE, CPE, XCCDF,

CVSS

SP800-82	CVE
SP800-86	CVE
SP800-94	CVE
SP800-115	CVE, CCE, CVSS, CWE
SP800-117	CVE, OVAL, CCE, CPE, XCCDF,

CVSS

SP800-126	CVE, OVAL, CCE, CPE, XCCDF,
-----------	-----------------------------

CVSS

NIST Interagency Reports:

NISTIR-7007	CVE
NISTIR-7275	CVE, OVAL, CCE, CPE, XCCDF, CVSS
NISTIR-7435	CVE, CVSS, CWE
NISTIR-7511	CVE, OVAL, CCE, CPE, XCCDF, CVSS
NISTIR-7517	CVE



FD



EMAP SWAAP

Industry Uptake

Manually review code after security education

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

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

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

Resources

- A Process for Performing Security Code Reviews, Michael Howard, IEEE Security & Privacy July/August 2006.
<http://msdn.microsoft.com/en-us/library/aa302937.aspx>
- .NET Framework Security — Code Review;
<http://msdn.microsoft.com/en-us/library/aa302937.aspx>
- **Common Weakness Enumeration, MITRE; <http://cwe.mitre.org/>**
- **Security Code Reviews;**
http://www.codesecurely.org/Wiki/view.aspx/Security_Code_Reviews
- Security Code Review — Use Visual Studio Bookmarks To Capture Security Findings; <http://blogs.msdn.com/alki/archive/2008/01/24/security-code-review-use-visual-studio-bookmarks-to-capture-security-findings.aspx>
- Security Code Review Guidelines, Adam Shostack;
<http://www.verber.com/mark/cs/security/code-review.html>
- OS WASP Top Ten; http://www.owasp.org/index.php/OWASP_Top_Ten_Project

10001
01111
10001
11110
10001

Testing

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

Fuzz testing

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

Resources

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

CWE
CAPEC



SAFECode
Software Assurance Program For Excellence in Code
Driving Security and Integrity



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

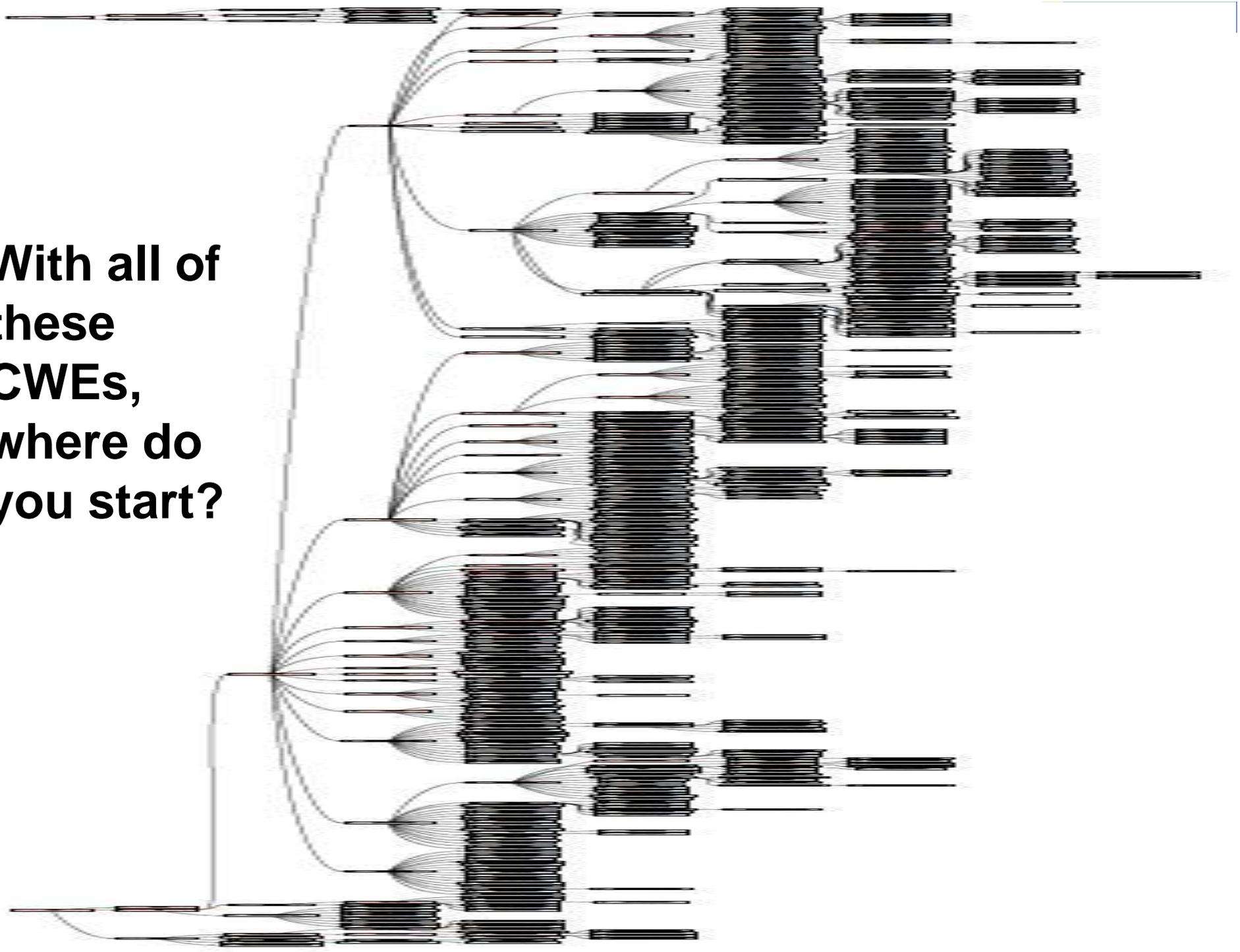
OCTOBER 8, 2008

LEAD WRITER Michael Howard, Microsoft Corp.

CONTRIBUTORS

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

**With all of
these
CWEs,
where do
you start?**



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, Cigital, Mandiant, Cigital, Secunia, Breach, SAIC, Aspect, WhiteHat
- Security Groups: OWASP, WASC

The screenshot shows the SANS AppSec Streetfighter Blog page. The header includes the SANS logo and navigation links: why SANS?, pick a course, why certify?, register now, and a search bar. Below the header is a banner for 'SANS FIRE 2010' in Baltimore, MD, from June 6-14, with a 'Click Here' button. The main content area is titled 'CWE/SANS TOP 25 Most Dangerous Programming Errors' and includes a sub-header 'What Errors Are Included in the Top 25 Programming Errors?'. It lists three categories: Insecure Interaction Between Components (8 errors), Risky Resource Management (10 errors), and Porous Defenses (7 errors). A 'Click on the headline in any of the listings for the MORE link' section lists various details provided for each error, such as ranking, remediation cost, and attack frequency. The page also features a 'Yearly Archive' for 2010 and 2009, and a 'Real Threats, Real Skills, Real Success' banner for the SANS Cyber Guardian Program.

Robert C. Seacord	CERT	Ryan Barnett	Breach
Pascal Meunier	CERIAS, Purdue University	Antonio Fontes	New Ac
Matt Bishop	University of California, Davis	Mark Elovuanti 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			
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](#)



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](#)

 **Homeland Security**

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](#)

2009

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.

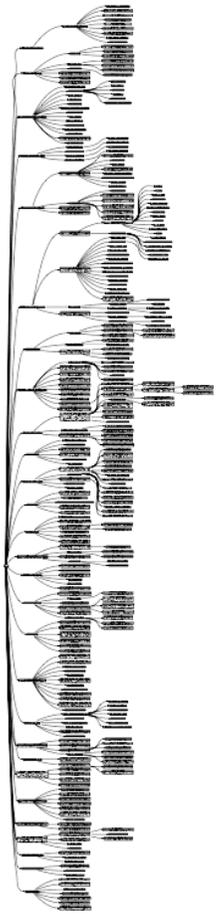
Rank	CWE ID	Name
[3]	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[7]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[12]	CWE-805	Buffer Access with Incorrect Length Value
[13]	CWE-754	Improper Check for Unusual or Exceptional Conditions
[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.

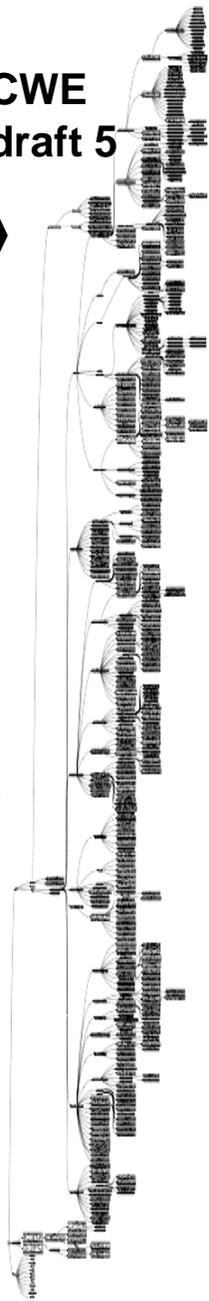
Rank	CWE ID	Name
[5]	CWE-285	Improper Access Control (Authorization)
[6]	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[10]	CWE-311	Missing Encryption of Sensitive Data
[11]	CWE-798	Use of Hard-coded Credentials
[19]	CWE-306	Missing Authentication for Critical Function
[21]	CWE-732	Incorrect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

**PLOVER
(CWE
draft 1)**



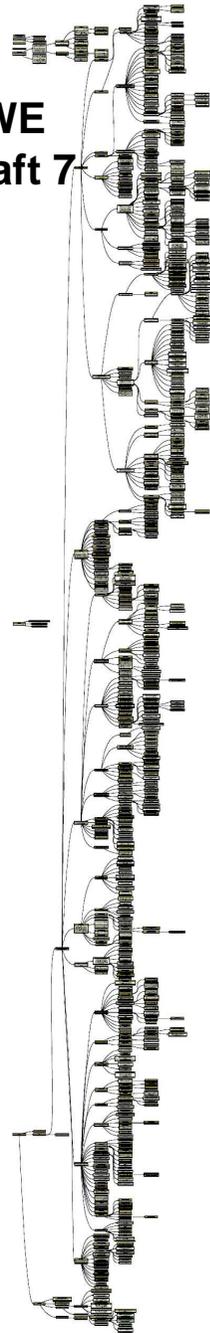
**2005
300 nodes**

**CWE
draft 5**



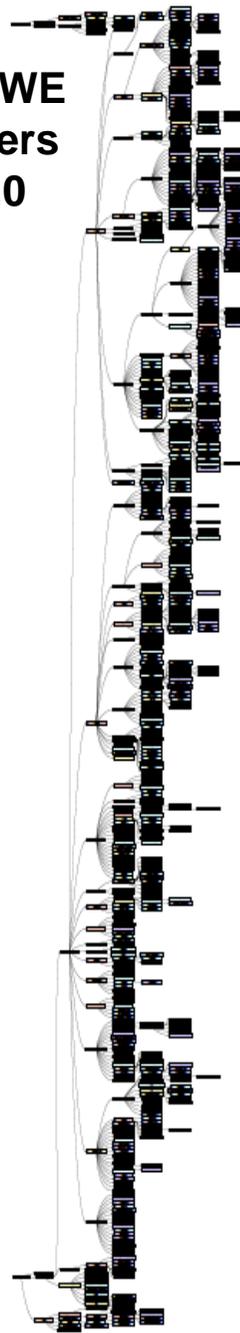
**2006
599 nodes**

**CWE
draft 7**



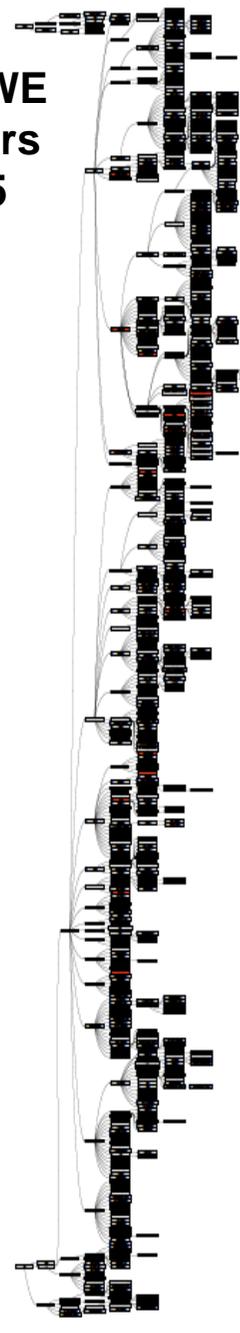
**2007
634 nodes**

**CWE
Vers
1.0**



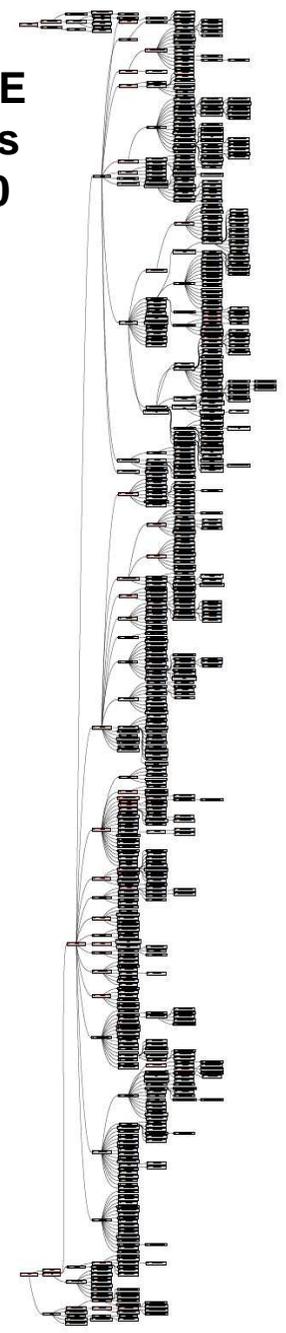
**2008
673 nodes**

**CWE
Vers
1.5**



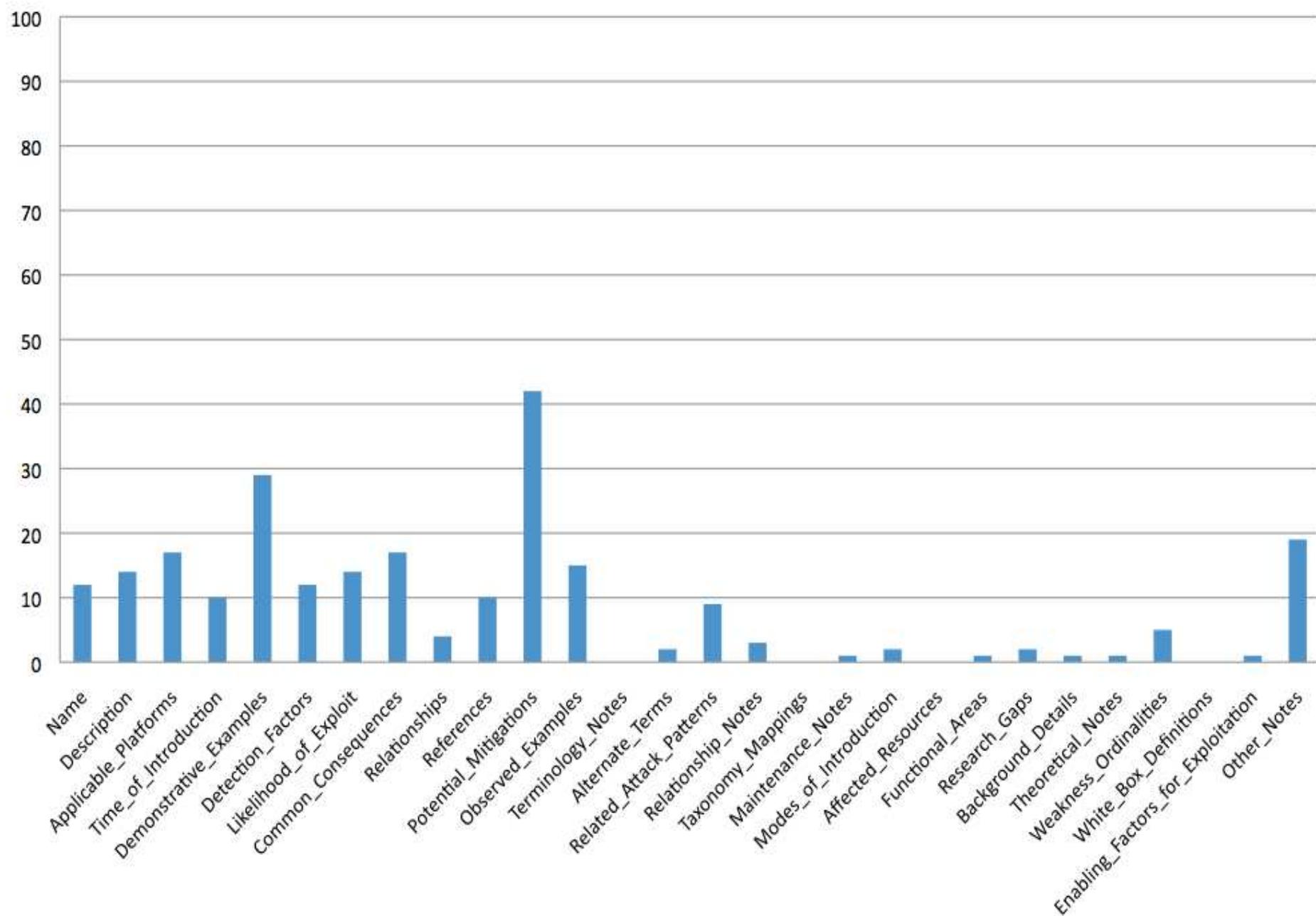
**2009
799 nodes**

**CWE
Vers
1.10**

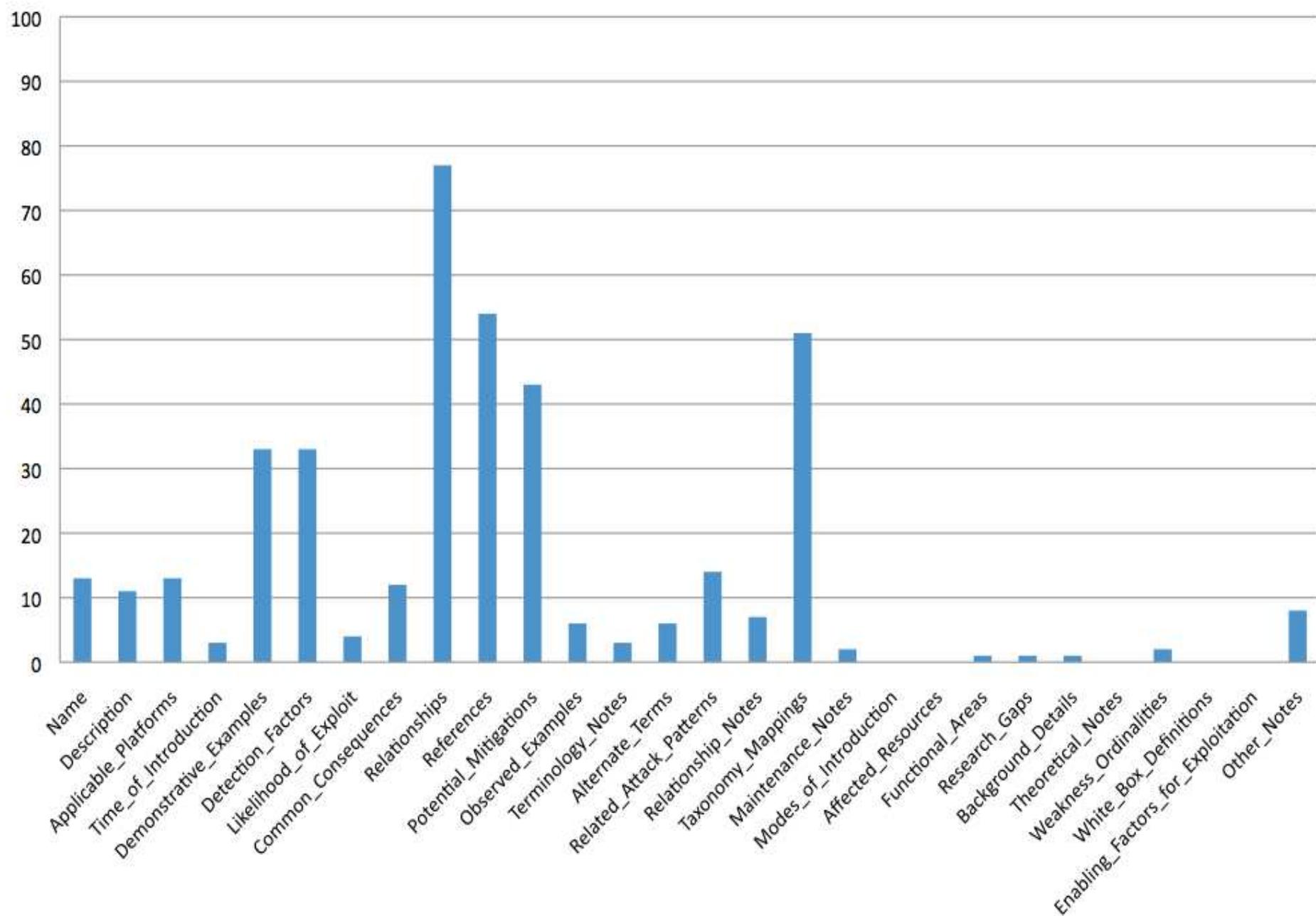


**Sep 2010
828 nodes**

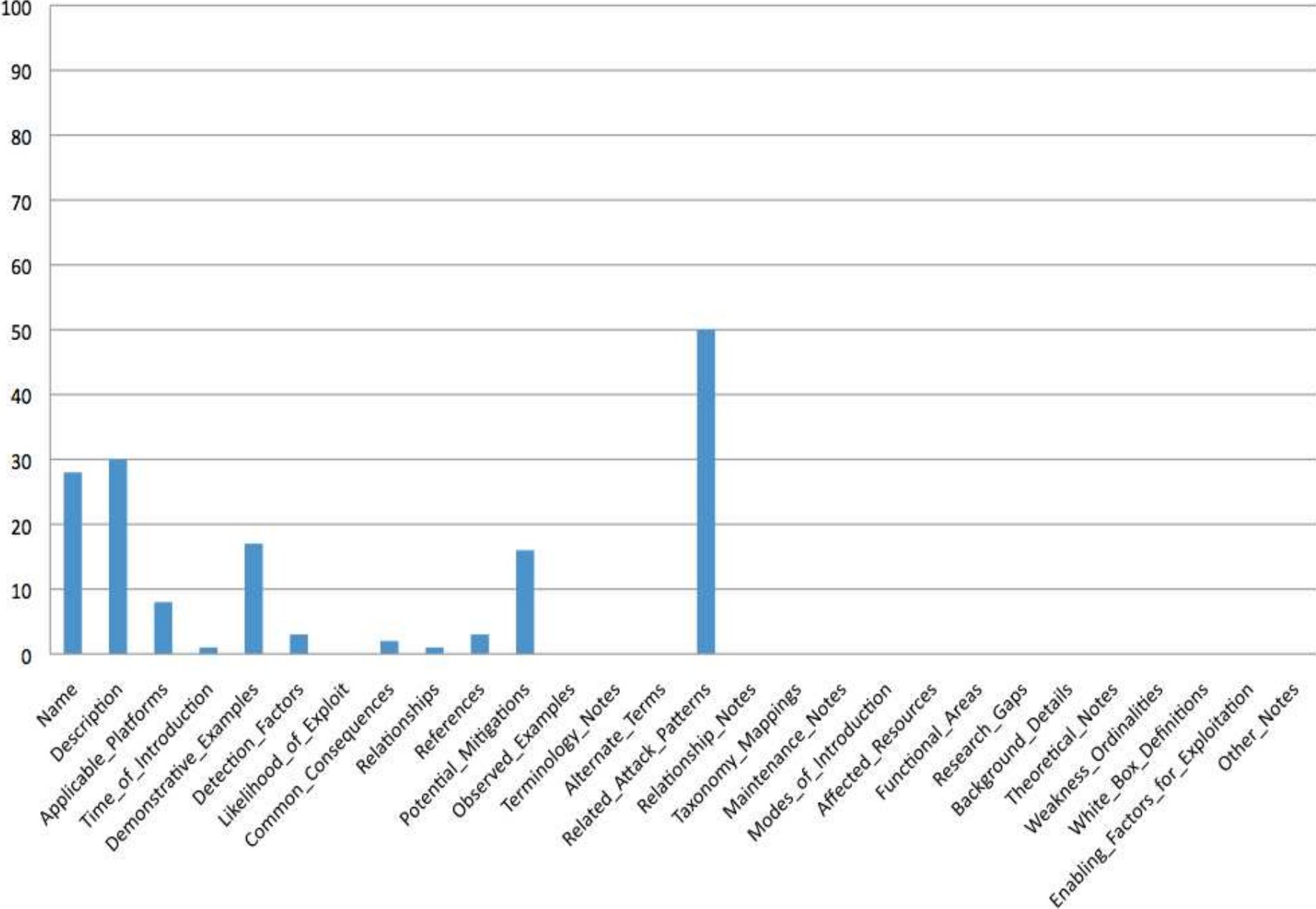
Field Changes - 1.6 to 1.7 (28 December 2009)



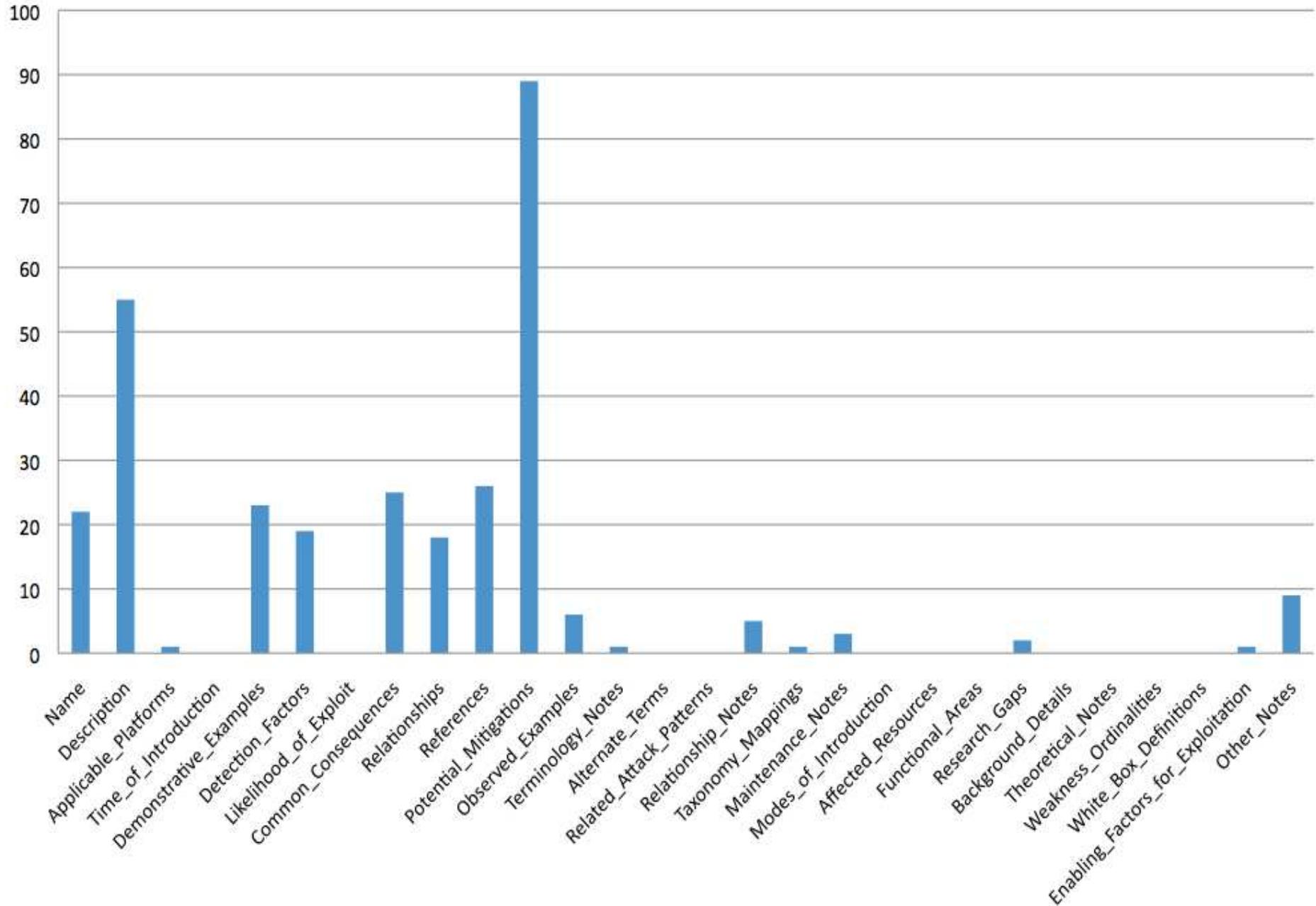
Field Changes - 1.7 to 1.8 (16 February 2010)



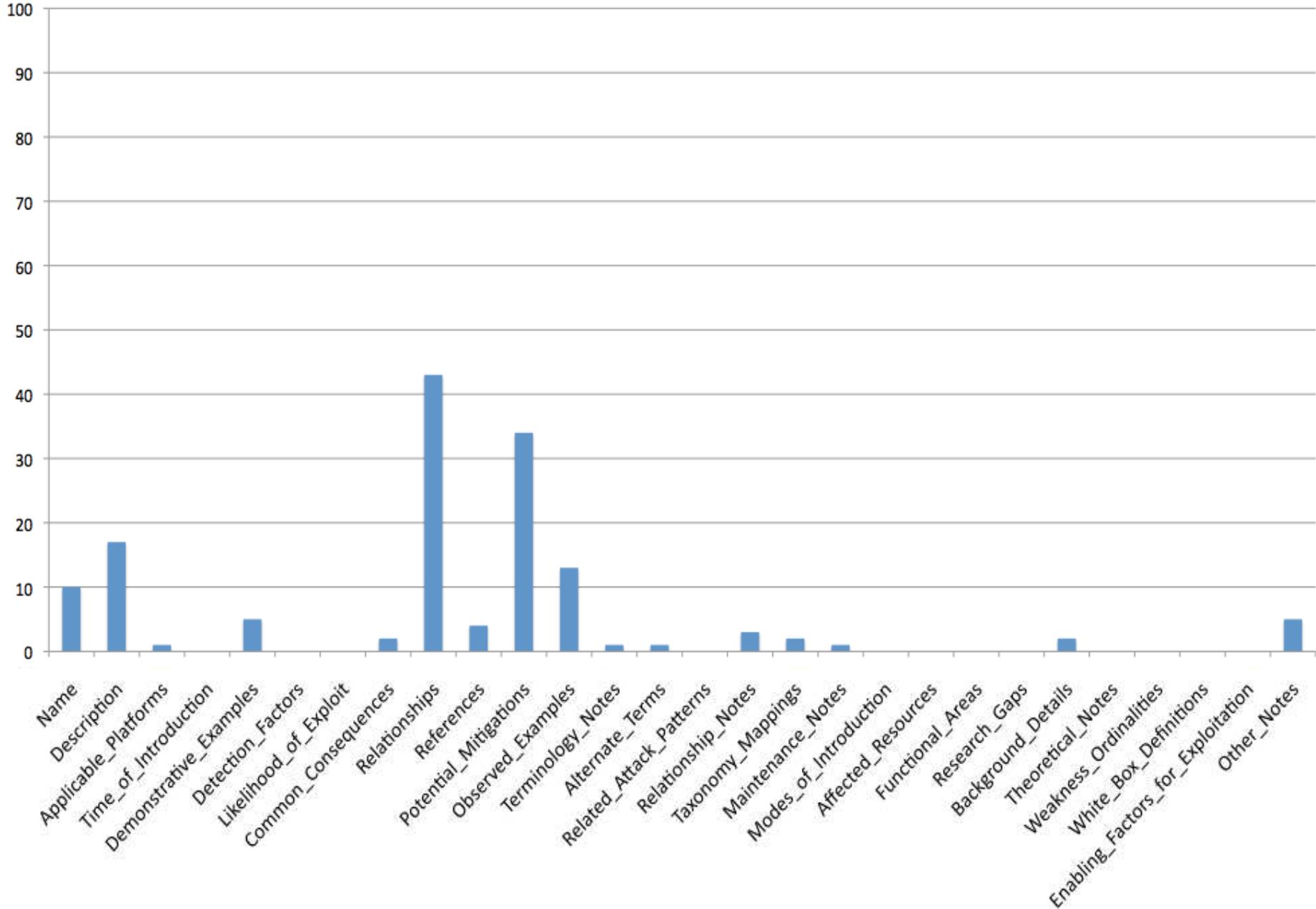
Field Changes - 1.8 to 1.8.1 (5 April 2010)



Field Changes - 1.8.1 to 1.9 (21 June 2010)



Field Changes - 1.9 to 1.10 (27 September 2010)



CWE version 1.10

Summary of Entry Types

Type	Version 1.9	v1.10
Category	119	119
Chain	3	3
Composite	6	6
Deprecated	11	11
View	24	24
Weakness	658	665

Nodes Added to v1.10

CWE-ID	CWE Name
820	Missing Synchronization
821	Incorrect Synchronization
822	Untrusted Pointer Dereference
823	Use of Out-of-range Pointer Offset
824	Access of Uninitialized Pointer
825	Expired Pointer Dereference
826	Premature Release of Resource During Expected Lifetime

CWE web site visitors by City





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 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 the SDL covers every one of 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 registry, in the case of Windows Vista and later, 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. Use 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 any 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. Be strict about what content or filename. As you view, look for or access file names, make sure the name is appropriate to valid data. Remember "known good" is a good way to a

CWE-426: Untrusted Old Versions

Old versions searched the root directory filenames, with problems if they had a weak prefix, weak or aren't completely guaranteed not to use searches or paths from a permitted source, environment, remedy is to internationalize terms—for example, Vista, the API doesn't exist in version of Windows named "program" (operating system) correct path to

CWE-94: Failure to Generate

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 more 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.

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 cookies, is subject to attack. It's important to protect it by using the appropriate control lists (ACLs) or permissions for persistent data and some 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 arbitrarily file data if they have the data that's used as part of a path name. It's critical

CWE-119: Failure to Constrain Memory Operations

The dreaded buffer overflow is a scourge of C and C++ and a vulnerability type that has more headaches than bugs. The best way to reduce the problem is to move away from C and C++ where it makes sense and use higher-level languages such as Ruby, C#, and Java because they don't offer direct access to memory. For C and C++, cautious developers should use "known bad" functions such as strcpy, strncpy, sprintf, and gets) and use secure versions. Visual C++ has many weak APIs at compile time and you should strive to use them. Also, fuzz testing and static analysis can help identify potential buffer overruns. operating-system-level tools such as address space layout randomization and no execution can help reduce the chance of buffer overruns. It's exploitable

CWE-642: External Control of Critical State

Unprotected state information, such as profile data or cookies, is subject to attack. It's important to protect it by using the appropriate control lists (ACLs) or permissions for persistent data and some 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 arbitrarily file data if they have the data that's used as part of a path name. It's critical

Basic Training

Editors: Richard Ford, rford@se.fi.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

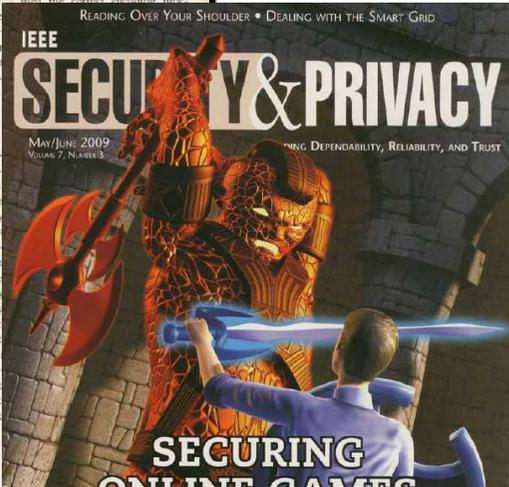
MICHAEL HOWARD
Microsoft

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, CWSy-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 reargue 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.



READING OVER YOUR SHOULDER • DEALING WITH THE SMART GRID

IEEE SECURITY & PRIVACY
May/June 2009
Volume 7, Number 3
ENSURING DEPENDABILITY, RELIABILITY, AND TRUST

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.

SECURING ONLINE GAMES

CWE-116: Improper Output Encoding

You could really use...

Basic Training

68

Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

MICHAEL HOWARD

Special thanks to Robert A. Wierle 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-ash on Failed Initialization
- Missing Initialization
- Incomplete Cleanup
- Improper Cleanup on Thrown Exception
- Improper Initialization (149)

Channel and Path Errors

- Channel Errors
- Failure to Protect Alternate Path
- Uncontrolled Search Path Element
- Unquoted Search Path on 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 Destruction
 - State Synchronization Error
 - Mutable Objects Passed by Reference
 - Pointing Mutable Objects to an Unchecked Method
 - External Control of Critical State Data (148)
- Session Fixation
- Concurrency Issues
- Temporary File Issues
- Covert Timing Channel
- Technology-Specific Time and State Issues
- Symbolic Name not Mapping to Correct Object
- Signal Errors
- Unrestricted Externally Accessible Lock
- Double-Checked Locking
- Insufficient Session Expiration
- Insufficient Synchronization
- Use of a Non-reentrant Function in an Unsynchronized Context
- Improper Control of a Resource Through its Lifetime
- Exposure of Resource to Wrong Sphere
- Incorrect Resource Transfer Between Spheres
- Use of a Resource after Expiration or Release
- External Influence of Sphere Definition
- Uncontrolled Recursion
- Redirect Without Path

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
- Often Missed Arguments and Parameters
- Uncaught Exception
- Escalation with Unnecessary Privileges (130)
- Often Missed String Management
- J2EE Bad Practices: Direct Use of Sockets
- Unchecked Return Value
- Failure to Change Working Directory in chroot jail
- Reliance on DNS Lookups in a Security Decision
- Failure to Follow Specification
- Failure to Provide Specified Functionality

Web Problems

- Failure to Sanitize CRLF Sequences in HTTP Headers (HTTP Response Splitting)
- Inconsistent Interpretation of HTTP Requests (HTTP Request Smuggling)
- Improper Sanitization of HTTP Headers for Scripting Syntax
- Use of Non-Canonical URI Paths for Authentication 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 (164)
- Empty Synchronized Block
- Exploit Call to Finalize
- Reachable Assertion
- Use of Potentially Dangerous Function

Credentials Management

- Hard-Coded Password (131)
- Unvetted 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 Signatures
- Use of Lost Trusted Source
- Acceptance of Extraneous Untrusted Data With Trusted Data
- Improperly Trusted Remote DNS
- Insufficient Type Distinction
- Over-Reliance on Parsers (136) (137)
- Failure to Add Integrity Check Value
- Improper Validation of Integrity Check Value
- Trust of System Event Data
- Reliance on File Name or Extension of Externally-Supplied File
- Reliance on Obfuscation or Obfuscation of Security-Relevant Inputs without Integrity Checking

Privacy Violation

- Reliance on Cookies without Validation and Integrity Checking
- Client-Side Enforcement of Server-Side Security (162)
- Improperly Implemented Security Check for Standard
- Improper Authentication
- User Interface Security Issues
- Use of Insufficiently Random Values (133)
- Logging of Excessive Data
- Certificate Issues

Insufficient Encapsulation

- Mobile Code Issues/Missing Custom Error Page
 - Public (or-writable) Method Without Final (Object Block)
 - Use of Inner Class Containing Sensitive Data
 - Critical Public Variable Without Final Modifier
 - Unvetted Use of Code Without Integrity Check (144)
 - Any Undeclared Public, Final, and Static
 - Analysis 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
 - Clear Text Transmission of Sensitive Information (138)
 - Sensitive Code in a HTTPS Session Without Secure Attributes
- Removable One-Time Pad
- Inadequate Encryption Strength
- Use of a Broken or Buggy Cryptographic Algorithm (135)
- Use of RSA Algorithms without OAEP

Permissions, Privileges, and Access Controls

- Access Control Mechanism Issues (134)
- Permission Issues
 - Incorrect Default Permissions
 - Incorrectly Inherited Permissions
 - Incorrectly Preserved Inherited Permissions
 - Incorrectly Executable Assigned Permissions
 - Improper Handling of Executable Permissions
 - Improper Preservation of Permissions
 - Exposed Unusable Access Method
 - Incorrect Permissions Assigned for Critical Resources (132)
 - Permission Race Condition During Resource Copy

Password in Configuration File

- Improper Ownership Management
- Unsecured User Management

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

Reliance on Package-level Scope

- J2EE Framework: Sensitive Unserializable Objects to Disk
- Deserialization of Untrusted Data
- Serializable Class Containing Sensitive Data
- Information Leak through Class Cloning
- Public Data Assigned to Private Array-Typed Field
- Private Array-Typed Field Returned from a Public Method
- Public Static Final Field References Mutable Object
- Exposed Dangerous Method or Function
- Critical Variable Declared Public
- Access to Critical Private Variable via Public Method

Data Handling

- Numeric Errors
 - Use of Incorrect Byte Ordering
 - Unchecked Array Indexing
 - Incorrect Conversion Between Numeric Types
 - Unquoted Sign Operator
 - Signed vs Unsigned Conversion Error
 - Unsigned to Signed Conversion Error
 - Numeric Truncation Error
 - Incorrect Calculations (142)
 - Incorrect Calculation of Buffer Size
 - Integer Overflow or Wraparound
 - Integer Underflow (Wrap or Wraparound)
 - Off by one Error
 - Divide by Zero
- Representation Errors
 - Cloning, Generalization, and Comparison Errors
 - Balance on Data Memory Layout
- Information Management Errors
 - Information Leak (Information Disclosure)
 - Information Leak Through Sort Data
 - Process Leak Through Data Sources
 - Dataquery Information Leak
 - Print Message Information Leak (140)
 - Cross-Boundary Crossing Information Leak
 - Iterated Information Leak
 - Process Environment Information Leak
 - Information Leak Through Debug Information
 - Specific Information Disclosed 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 GET Request
 - Information Leak Through Indexing of Private Data
 - Information Loss or Distortion
 - Combination Error, Data Loss 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 Syntactically Invalid Structure
- Modification of Assumed-Immutable Data (MAID)
 - Improper Input Validation (120)
 - Pathnames Traversal and Escape Sequence Errors
 - Process Control
 - Missing EOL Validation
 - Failure to Sanitize Data into a Different Plane (Typecast)
 - Improper Sanitization of Special Elements used in a Command ("Command Injection") (117)
 - Failure to Preserve Web Page Structure ("Class-File Scripting") (111)
 - Improper Sanitization of Special Elements used in an XML Document ("XML Injection") (116)
 - Failure to Sanitize Data into LAMP Directs ("LAMP Injection")
 - XML Injection (aka Blind XPath Injection)
 - Failure to Sanitize CRLF Sequences ("CRLF Injection")
 - Uncontrolled Format String
 - Failure to Sanitize Special Elements into a Different Plane
 - Argument Injection or Modification
 - Improper Control of Resource Identification ("Resource Injection")
 - Failure to Control Generation of Code ("Code Injection") (141)
 - Improper Sanitization of Special Elements
 - Technology Specific Input Validation Problems
 - Misinterpretation of Input
 - Unchecked Input for Loop Condition
 - Null Byte Injection Error (Hex or Null Byte)
 - Direct Use of Unsafe J2E
 - Improper Output Sanitization for Logs
 - Failure to Control Operations within the Bounds of a Memory Buffer (118)
 - Use of Externally-Controlled Input to Select Classes or Code ("Class's Reflection")
 - ASP.NET Misconfigurations: Not Using Input Validation Framework
 - URL Redirection to Untrusted Site ("Open Redirect")
 - Variable Extraction Errors
 - Unvetted Function Hook Arguments
 - External Control of File Name or Path (119)
 - Improper Address Initialization in IOCTL, with METHOD, METHOD IO Control Code
 - Use of Path Manipulation Function with a Maximum-sized Buffer

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

John

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.

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.



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

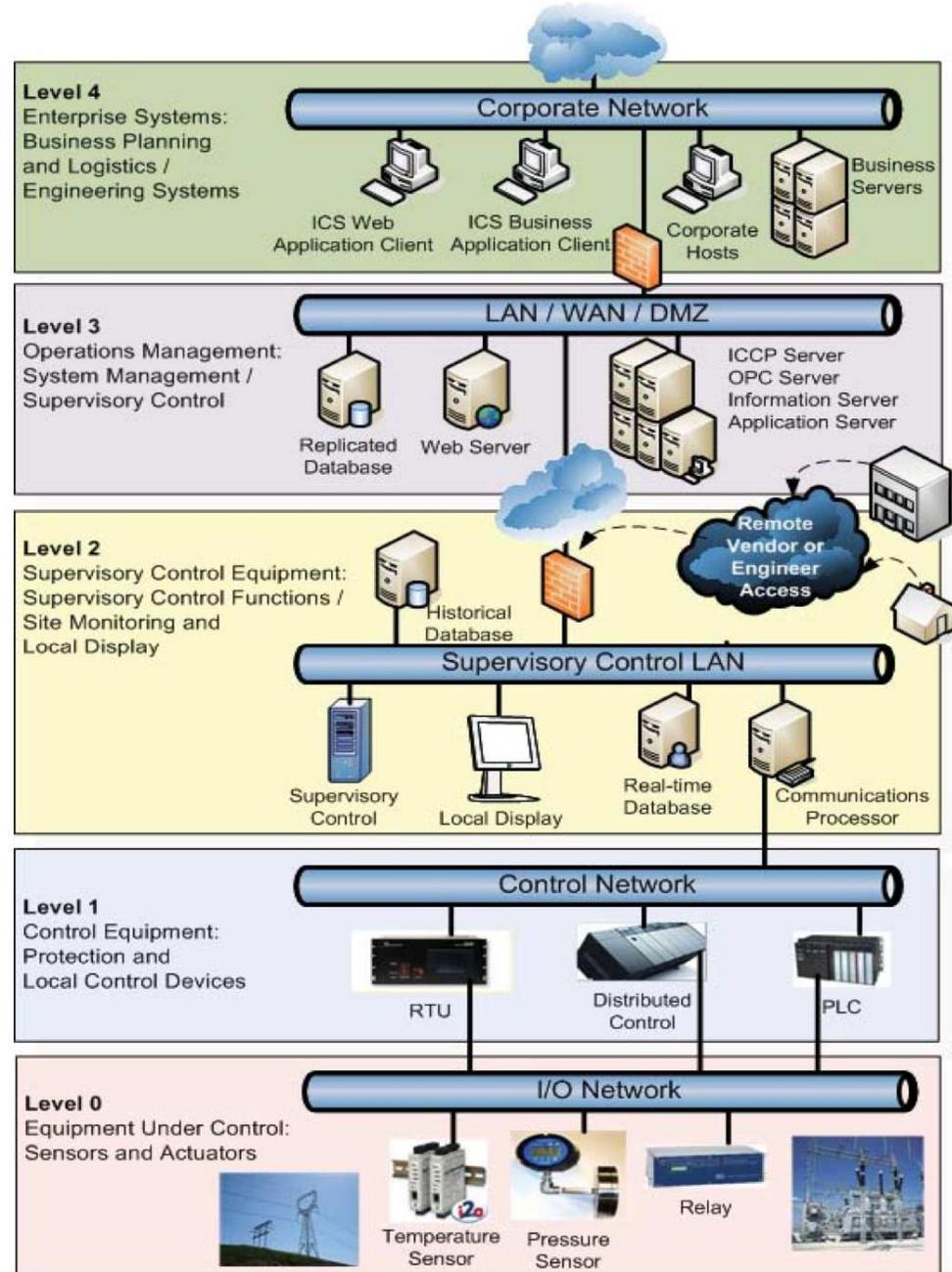
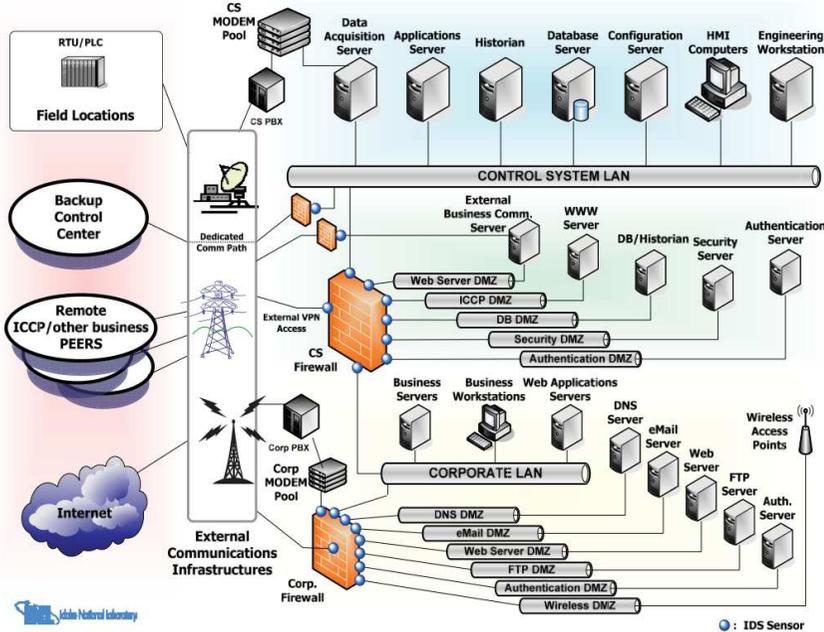


Table 27. Most common programming errors found in ICS code.

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

Linkage with Fundamental Changes in Enterprise Security Initiatives

Twenty Critical Controls for Effective Cyber Defense Guidelines

What the 20 CSC Critics say...

20 Critical Security Controls - Version 2.0

- [20 Critical Security Controls - Introduction \(Version 2.0\)](#)
- [Critical Control 1: Inventory of Authorized and Unauthorized Devices](#)
- [Critical Control 2: Inventory of Authorized and Unauthorized Applications](#)
- [Critical Control 3: Secure Configurations for Hardware and Software](#)
- [Critical Control 4: Secure Configurations for Network Devices](#)
- [Critical Control 5: Boundary Defense](#)
- [Critical Control 6: Maintenance, Monitoring, and Analysis of Security Alerts](#)
- [Critical Control 7: Application Software Security](#)
- [Critical Control 8: Controlled Use of Administrative Privileges](#)
- [Critical Control 9: Controlled Access Based on Need to Know](#)
- [Critical Control 10: Data Protection](#)
- [Critical Control 11: Incident Response and Computer Forensics](#)
- [Critical Control 12: Business Continuity and Disaster Recovery](#)
- [Critical Control 13: Multi-Factor Authentication](#)
- [Critical Control 14: Security Awareness and Training](#)
- [Critical Control 15: Vendor Security Assessments](#)
- [Critical Control 16: Information Security Policies and Procedures](#)
- [Critical Control 17: Penetration Testing](#)
- [Critical Control 18: Security Assessments](#)
- [Critical Control 19: Risk Management](#)
- [Critical Control 20: Security Incident Response and Computer Forensics](#)

Procedures and tools for implementing this control

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

CAG: Critical Control 7: Application Software Security

<< previous control

Consensus Audit Guidelines

next control >>

How do attackers exploit the lack of this control?

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

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






ISO/IEC JTC 1/SC 27 NXXXX
ISO/IEC JTC 1/SC 27/WG x NXXXXX

REPLACES: N

ISO/IEC JTC 1/SC 27
Information technology - Security techniques
 Secretariat: DIN, Germany

DOC TYPE: NB MWI Proposal for a technical report (TR)
TITLE: National Body New Work Item Proposal on "Secure software development and evaluation under ISO/IEC 15408 and ISO/IEC 18405"
SOURCE: INCITSACS 1, National Body of (JS)
DATE: 2008-09-30
PROJECT: 15408 and 18405
STATUS: This document is circulated for consideration at the forthcoming meeting of SC 27/WG 3 to be held in Redmond (WA, USA) on 2nd - 6th November 2008.
ACTION ID: ACT
DUE DATE:
DISTRIBUTION: P, D and L-Members
 W: Larry Sid 27 Chairman
 M: De Soete, SC 27 Vice-Chair
 E: J. Humphreys, K. Naraenara, Y. Bultin, M.-C. Kang, K. Rammberg, WG-Consensus
MEDIUM: Live Intra-server
NO. OF PAGES: xx

Secretariat ISO/IEC JTC 1/SC 27 -
 DIN Document Institute für Normung e. V., Burgstrasse 6, 10772 Berlin, Germany
 Telephone: +49 30 201-15952; Facsimile: +49 30 2501-7233; E-mail: iso@iso.din.de
[HTTP://www.iso/iso/secretariat/](http://www.iso/iso/secretariat/)

Common Criteria v4 CCDB
•TOE to leverage CAPEC & CWE
•Also investigating how to leverage ISO/IEC 15026
NIAP Evaluation Scheme
•Above plus
•Also investigating how to leverage SCAP

New Work Item Proposal
NP submitting
PROPOSAL FOR A NEW WORK ITEM

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

A proposal for a new work item shall be submitted to the secretariat of the ISO/IEC joint technical committee concerned with a copy to the ISO Central Secretariat.
Presentation of the proposal
Title: Secure software development and evaluation under ISO/IEC 15408 and ISO/IEC 18405

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

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

- A refinement of the IS 15408 Attack Potential evaluation table for software, taking into account the entries contained in the CAPEC and their characteristic.
- How the information for mitigating software common attack patterns and related weaknesses is used in an IS 15408 evaluation, in particular providing guidance on how to determine which attack patterns and weaknesses are applicable to the TOE, taking into consideration of:
 - the TOE technology;
 - the TOE security problem definition;
 - the interfaces the TOE exports that can be used by potential attackers;
 - the Attack Potentials that the TOE needs to provide resistance for.
- How the CAPEC and related Common Weakness Enumeration (CWE) weaknesses are used by the evaluator, who needs to consider at the applicant's attack patterns and be able to exploit specific related software weaknesses while performing the subsequent vulnerability analysis (AVA_VAN) activities on the TOE.
- How incomplete entries from the CAPEC are resolved during an IS 15408 evaluation.
- How the evaluator's attack and weakness analysis of the TOE incorporates other attacks and weaknesses not yet documented in the CAPEC.

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

The image features a blue gradient background. In the center, there are dark silhouettes of a dog and a cat. The dog is on the left, facing right, with its tail curved upwards. The cat is on the right, facing left, with its tail curved downwards. The word "Questions?" is written in white, bold, sans-serif font across the middle of the image, overlapping the silhouettes.

Questions?

ramartin@mitre.org