Status of the ITU-T CWE Cybersecurity Recommendation

Robert A. Martin

3 February 2011

ITU Overview

191 Member States

+700 Sector Members

ITU

Helping the World Communicate

ITU-T

Telecommunication standardization of network and service aspects

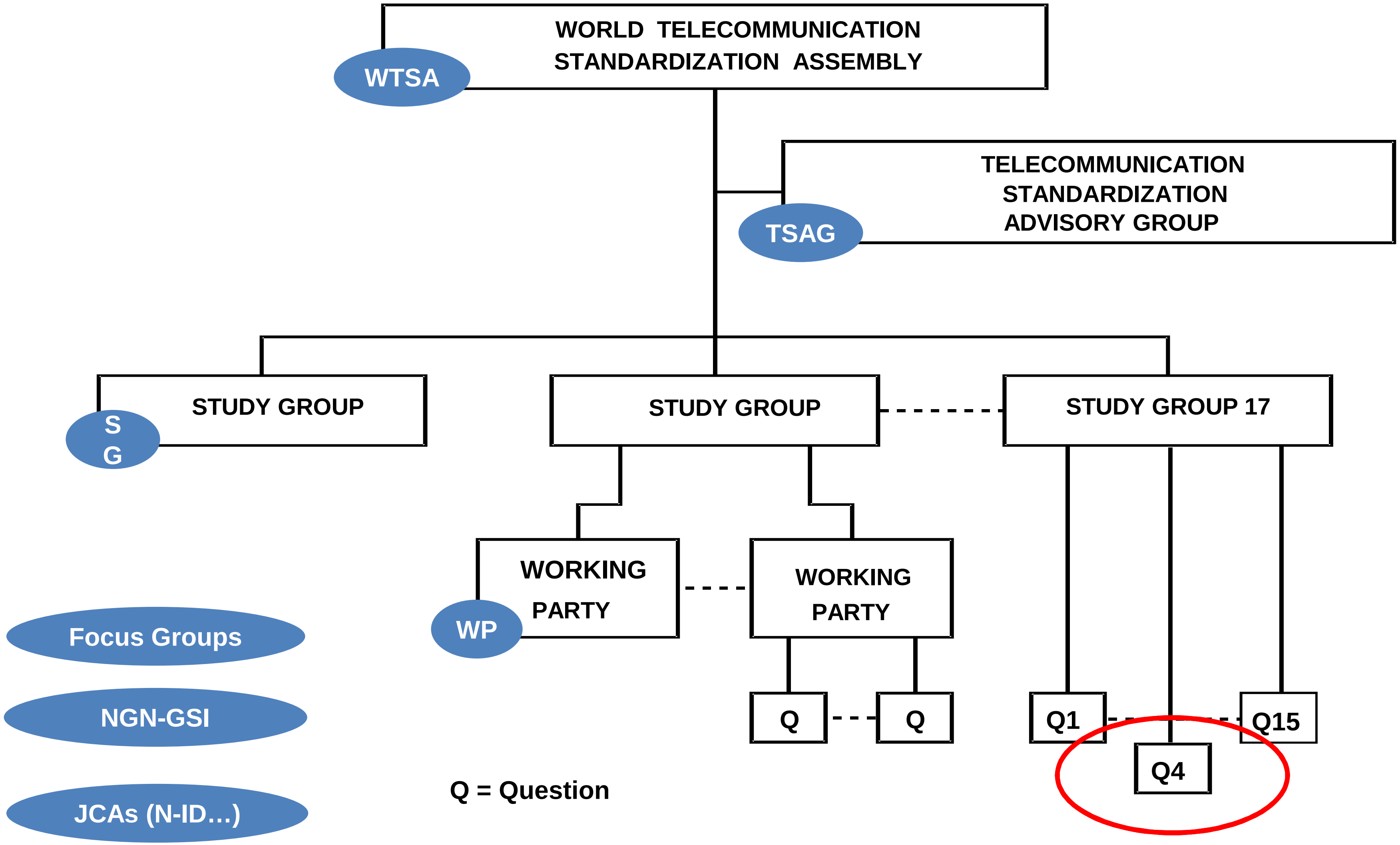
ITU-D

Assisting implementation and operation of telecommunications in developing countries

ITU-R

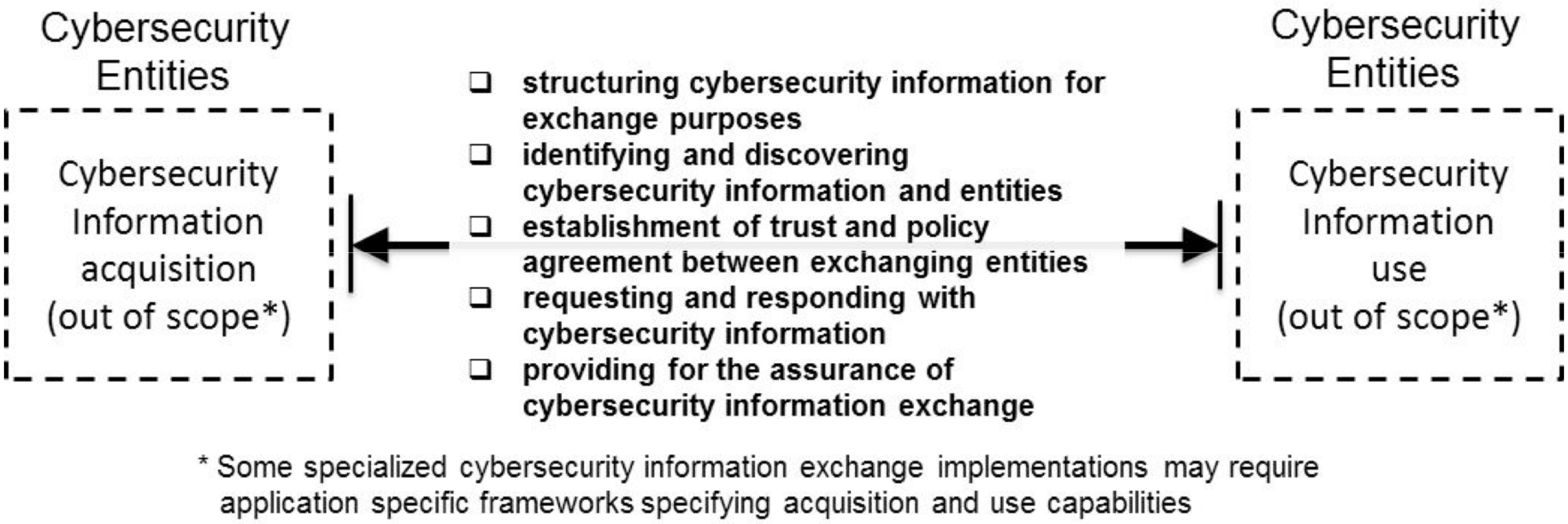
Radiocommunication standardization and global radio spectrum management

ITU-T structure



Q = Question

The CYBEX Model



Cybersecurity Entities

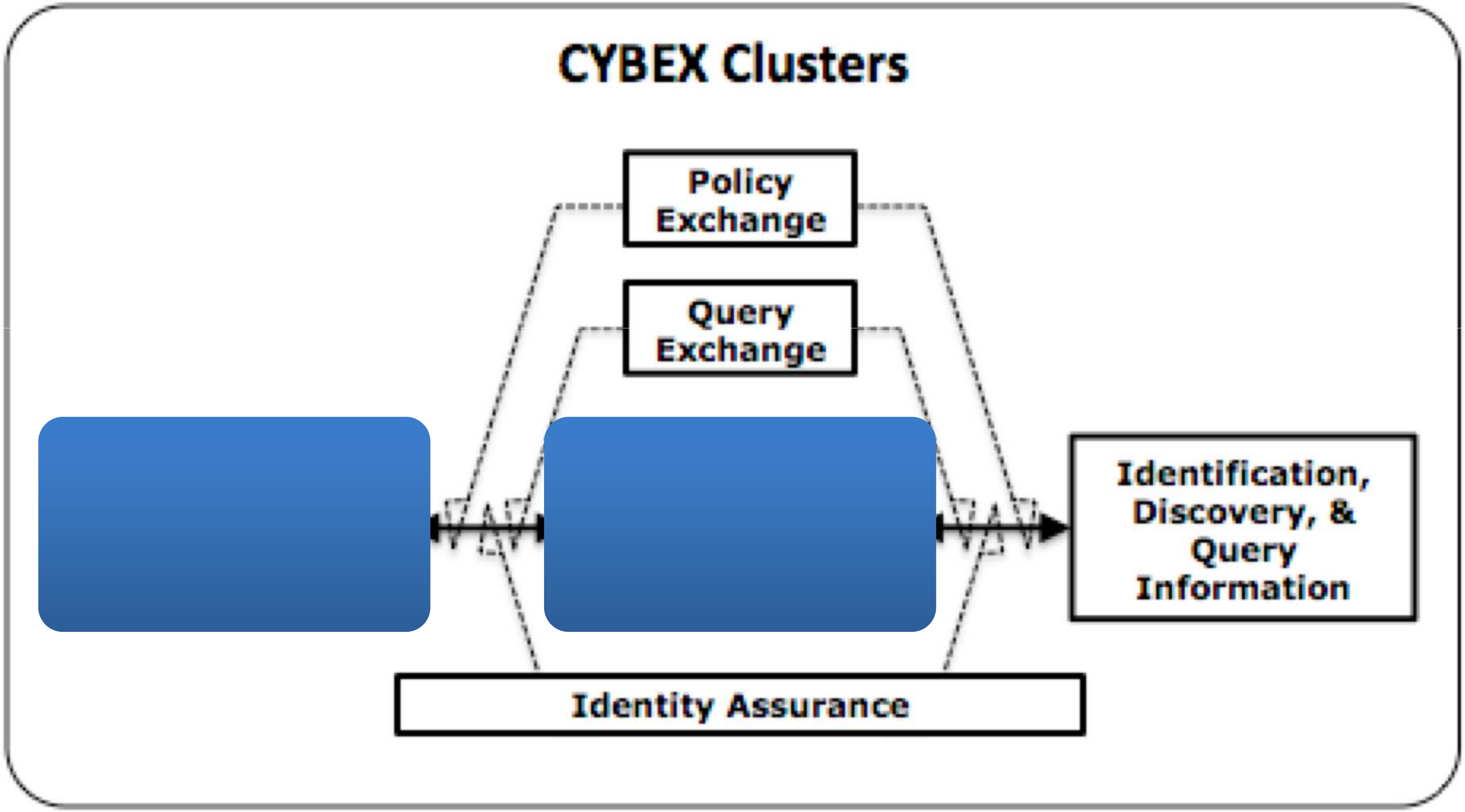
Cybersecurity Information acquisition (out of scope*)

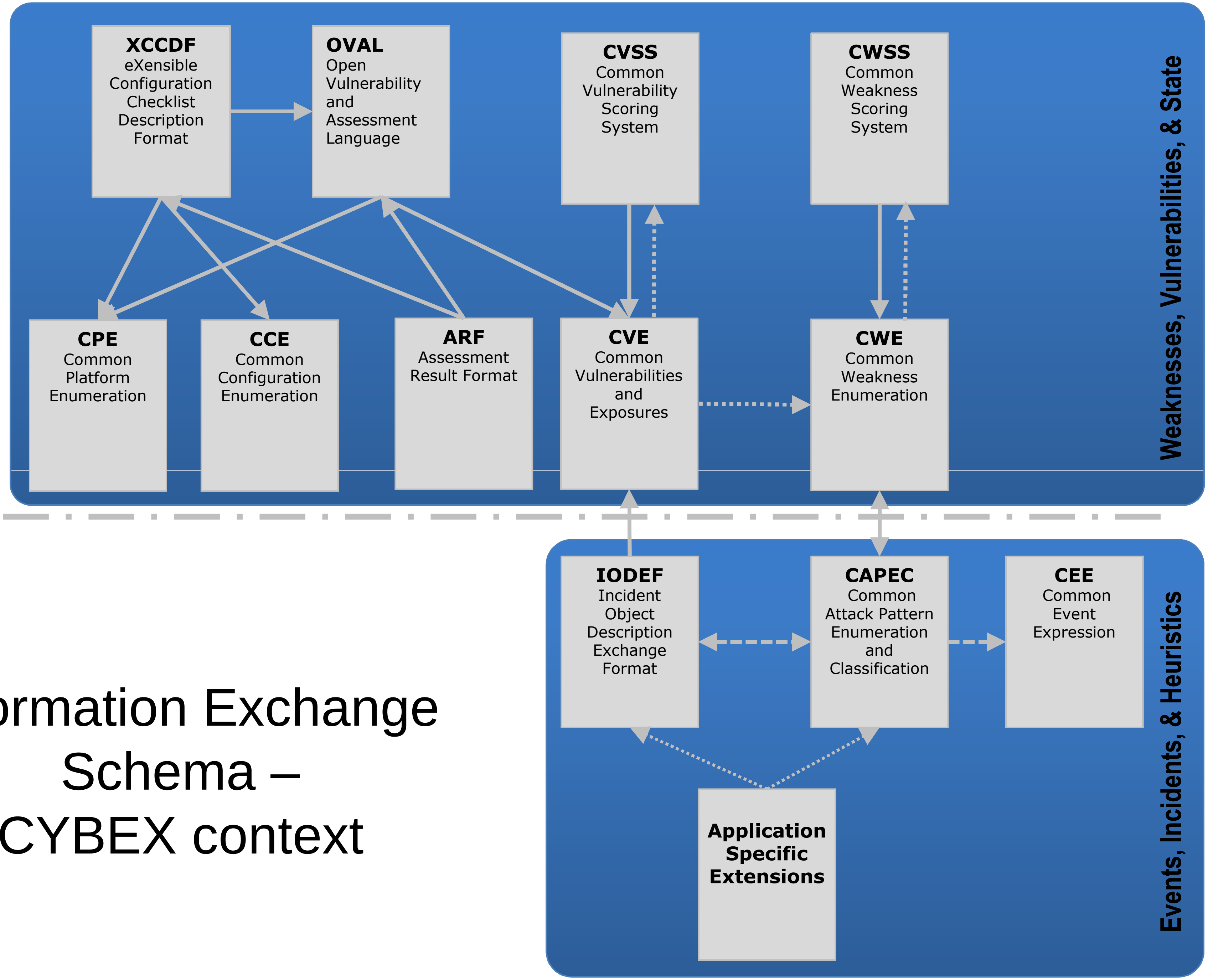
- ❑ structuring cybersecurity information for exchange purposes
- ❑ identifying and discovering cybersecurity information and entities
- ❑ establishment of trust and policy agreement between exchanging entities
- ❑ requesting and responding with cybersecurity information
- ❑ providing for the assurance of cybersecurity information exchange

Cybersecurity Entities

Cybersecurity Information use (out of scope*)

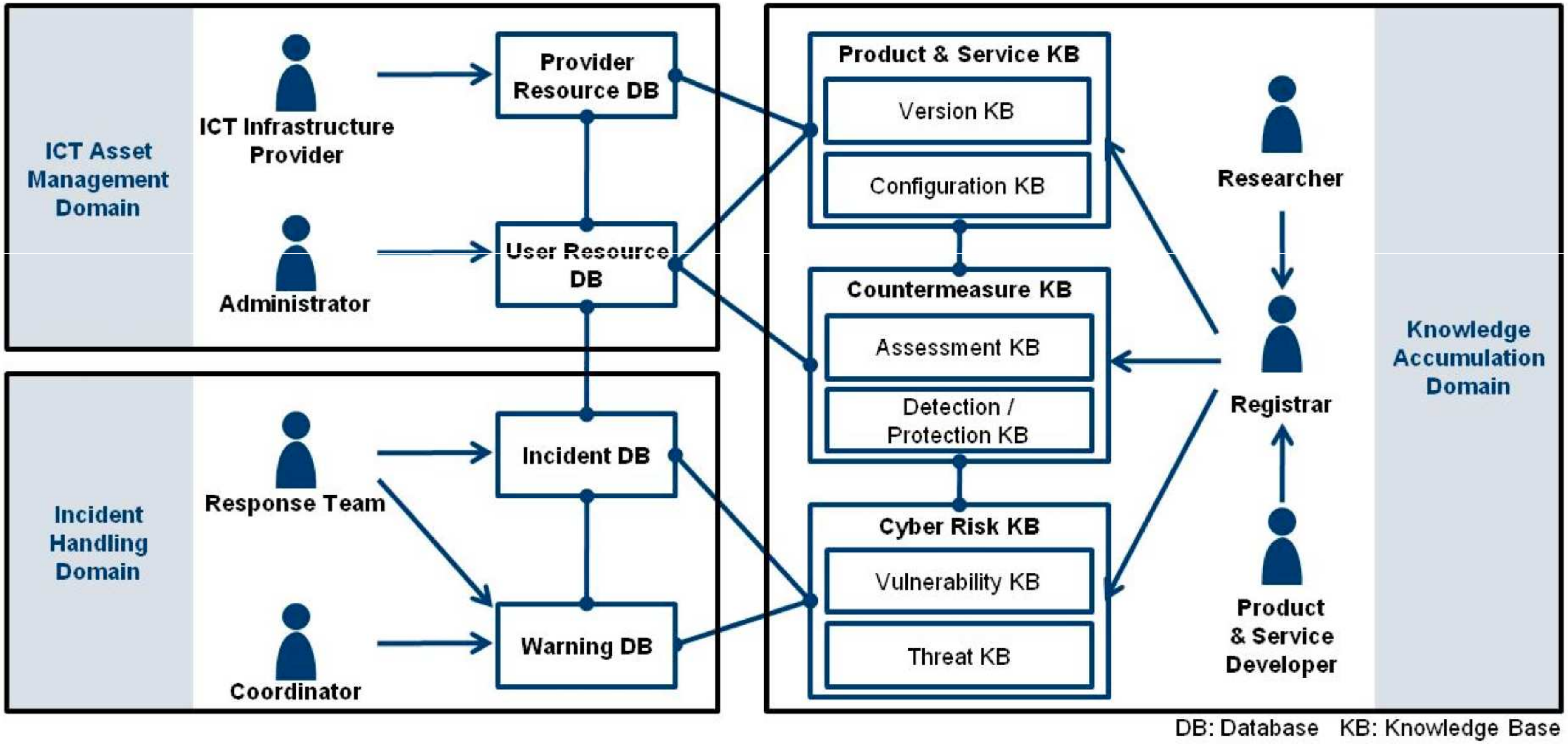
* Some specialized cybersecurity information exchange implementations may require application specific frameworks specifying acquisition and use capabilities


CYBEX Clusters
Policy Exchange
Query Exchange
Identification, Discovery, & Query Information
Identity Assurance

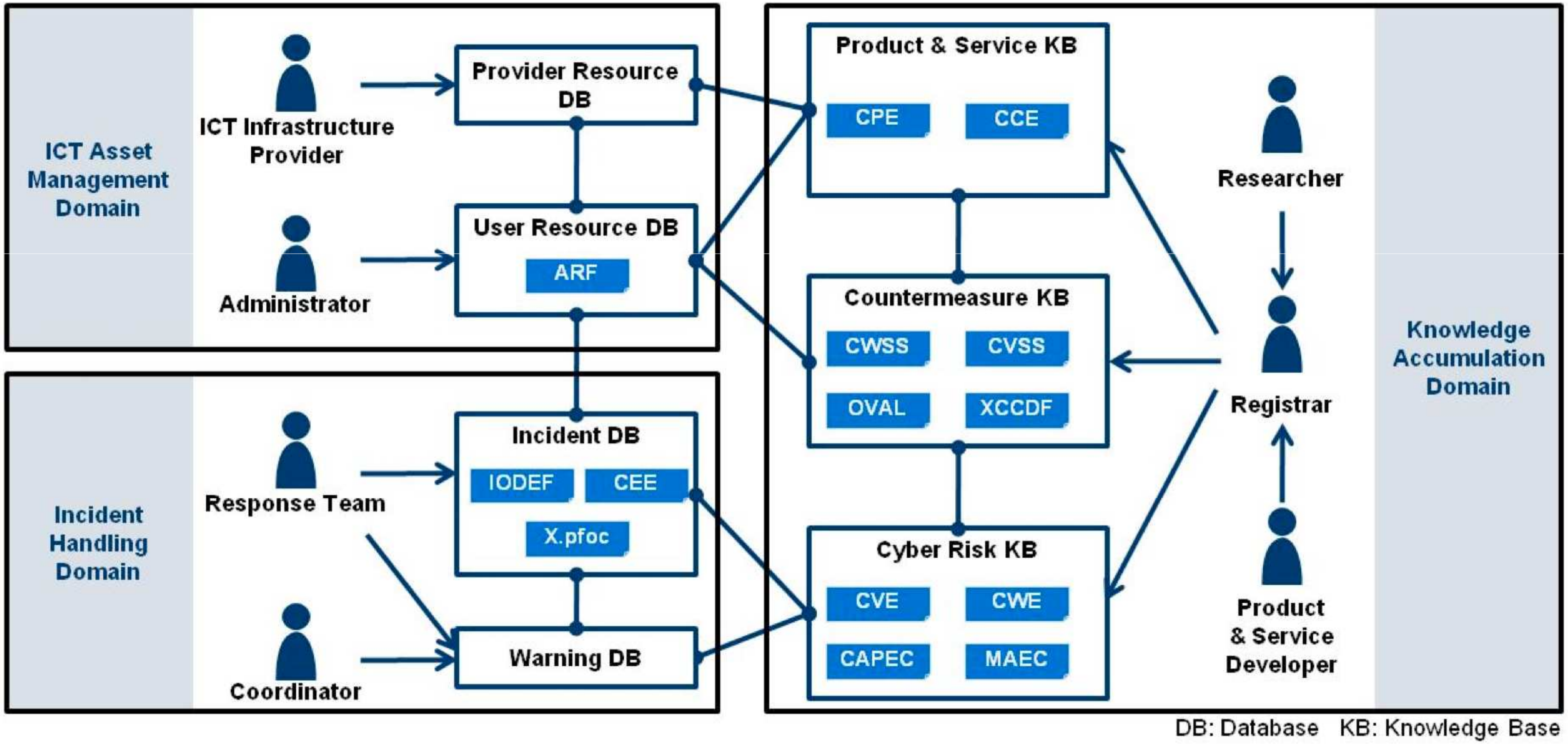

XCCDF
eXensible Configuration Checklist Description Format
OVAL
Open Vulnerability and Assessment Language
CVSS
Common Vulnerability Scoring System
CWSS
Common Weakness Scoring System
CPE
Common Platform Enumeration
CCE
Common Configuration Enumeration
ARF
Assessment Result Format
CVE
Common Vulnerabilities and Exposures
CWE
Common Weakness Enumeration
Weaknesses, Vulnerabilities, & State
IODEF
Incident Object Description Exchange Format
CAPEC
Common Attack Pattern Enumeration and Classification
CEE
Common Event Expression
Application Specific Extensions
Events, Incidents, & Heuristics

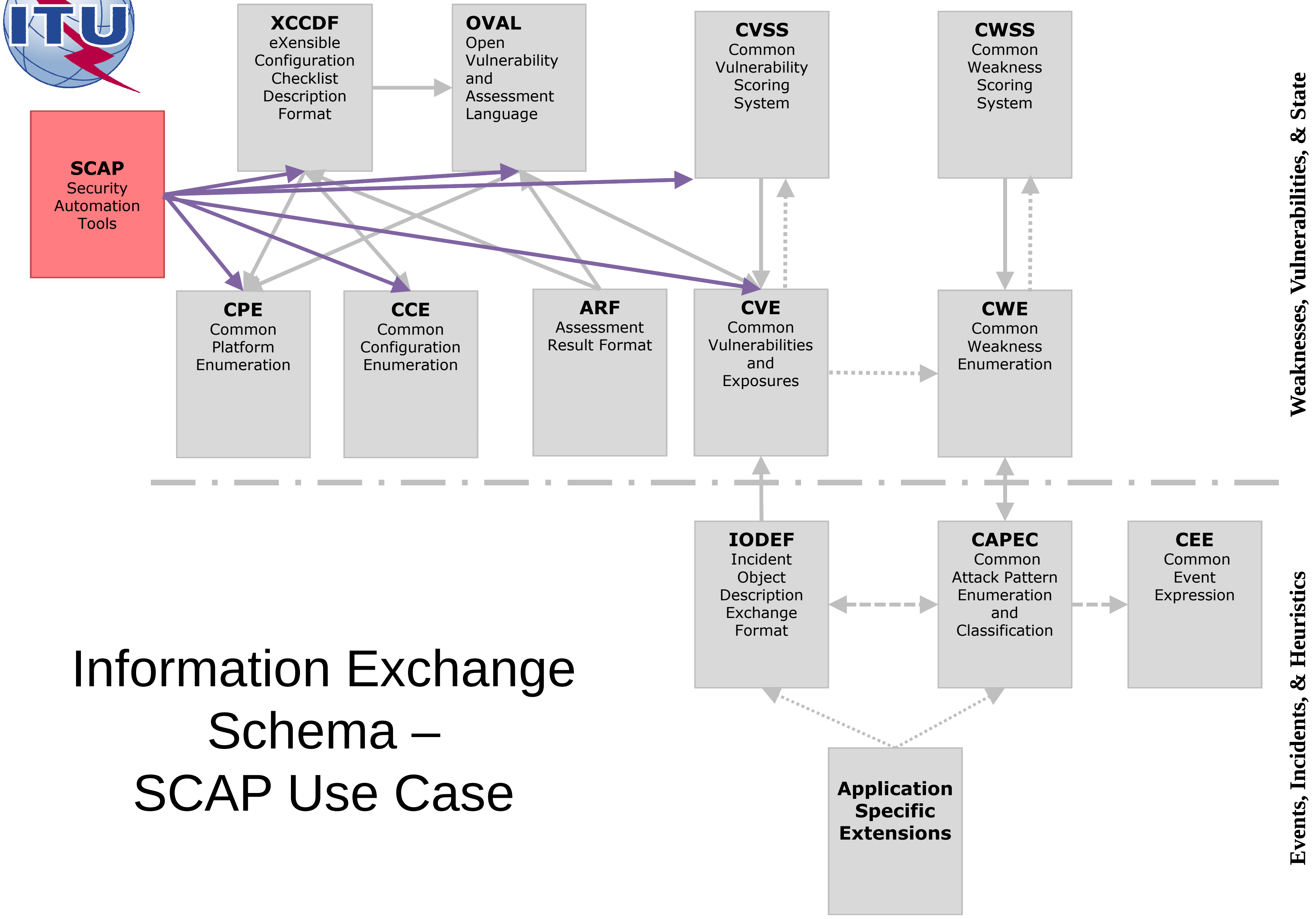
Information Exchange Schema – CYBEX context

CYBEX ontology model



Detailed view of the CYBEX ontology model with techniques shown




XCCDF
eXensible Configuration Checklist Description Format
OVAL
Open Vulnerability and Assessment Language
CVSS
Common Vulnerability Scoring System
CWSS
Common Weakness Scoring System
SCAP
Security Automation Tools
CPE
Common Platform Enumeration
CCE
Common Configuration Enumeration
ARF
Assessment Result Format
CVE
Common Vulnerabilities and Exposures
CWE
Common Weakness Enumeration
Weaknesses, Vulnerabilities, & State
IODEF
Incident Object Description Exchange Format
CAPEC
Common Attack Pattern Enumeration and Classification
CEE
Common Event Expression
Application Specific Extensions
Events, Incidents, & Heuristics

Information Exchange Schema – SCAP Use Case

ITU-T Study Group 17 Question 4:
Adopting the Information Security Community's Efforts

XXX is one of a class of ITU-T Recommendations that comes from a large, existing, global development and user community that has written and evolved an open specification that is made available to the ITU-T for adoption with agreement that any changes or updates to the specification will be done in a manner that ensures full technical equivalency and compatibility will be maintained, that discussions about changes and enhancements will be done through the original user community processes, and includes explicit reference to the corresponding specific version maintained by the user community. Thus, at the time of initial adoption of Rec. X.XXXX, a due diligence verification and statement of equivalency will occur; and as changes are effected among the user community, timely reflection of those changes will be incorporated in subsequent versions of the Recommendation through continued collaboration.

Status of ITU-T Recommendations

x-series	Title	ITU-T Status	Planned Determination
x.1500	Cybersecurity Information Exchange (CYBEX) Techniques	Final	Dec 2010
x.1520	Common Vulnerabilities and Exposures	Final	Dec 2010
x.1521	Common Vulnerability Scoring System	Final	Dec 2010
x.cwe	Common Weakness Enumeration	Draft	Aug 2011
x.oval	Open Vulnerability and Assessment Language	Draft	Aug 2011
x.cce	Common Configuration Enumeration	Draft	Aug 2011
x.capec	Common Attack Pattern Enumeration and Classification	Draft	Feb 2012
x.maec	Malware Attribute Enumeration and Classification	Draft	2012
x.cwss	Common Weakness Scoring System	Draft	2012
x.cee	Common Event Expression	Draft	2012
x.cpe	Common Platform Enumeration	Draft	2012
x.arf	Asset Reporting Format	Draft	2012
x.xccdf	Extensible Configuration Checklist Description Format	Draft	2012

Bob Martin, 3 March 2011

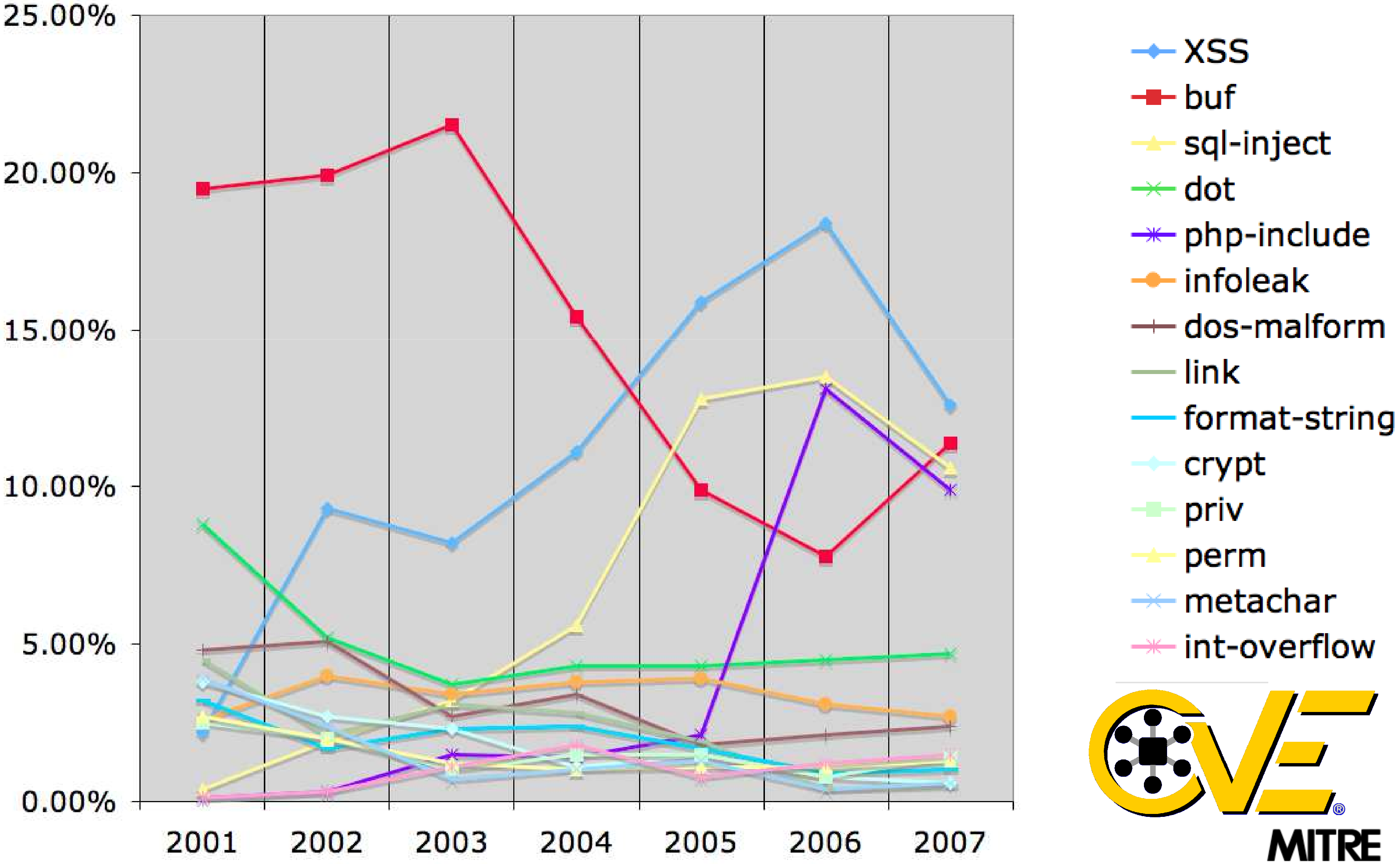
Mitigating the Top 25 Egregious Software Errors

Robert A. Martin

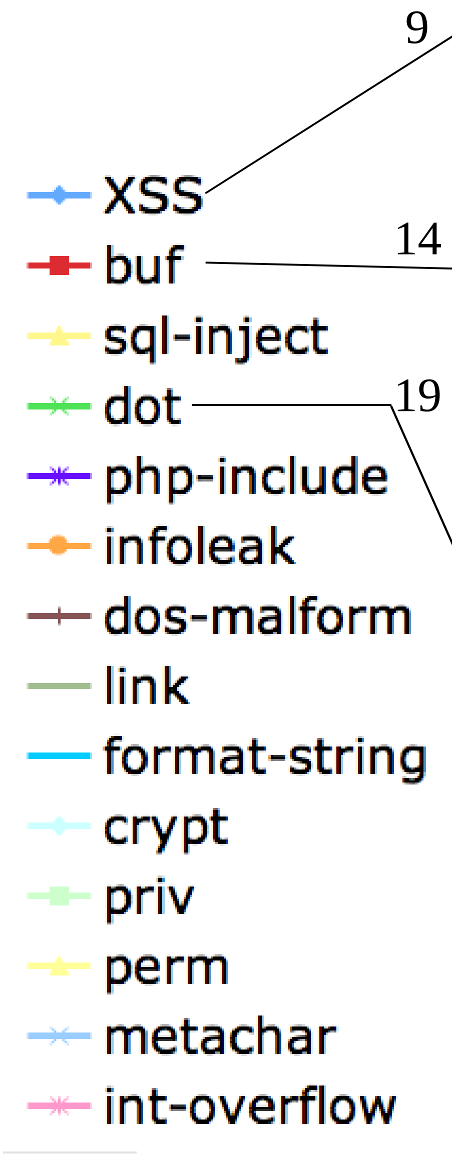
3 March 2011

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

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



Removing and Preventing the Vulnerabilities Requires More Specific Definitions…CWEs



Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (79)

- Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (80)
- Improper Neutralization of Script in an Error Message Web Page (81)
- Improper Neutralization of Script in Attributes of IMG Tags in a Web Page (82)
- Improper Neutralization of Script in Attributes in a Web Page (83)
- Improper Neutralization of Encoded URI Schemes in a Web Page (84)
- Doubled Character XSS Manipulations (85)
- Improper Neutralization of Invalid Characters in Identifiers in Web Pages (86)
- Improper Neutralization of Alternate XSS Syntax (87)

Improper Restriction of Operations within the Bounds of a Memory Buffer (119)

- Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (120)
- Write-what-where Condition (123)
- Out-of-bounds Read (125)
- Improper Handling of Length Parameter Inconsistency (130)
- Improper Validation of Array Index (129)
- Return of Pointer Value Outside of Expected Range (466)
- Access of Memory Location Before Start of Buffer (786)
- Access of Memory Location After End of Buffer (788)
- Buffer Access with Incorrect Length Value 805
- Untrusted Pointer Dereference (822)
- Use of Out-of-range Pointer Offset (823)
- Access of Uninitialized Pointer (824)
- Expired Pointer Dereference (825)

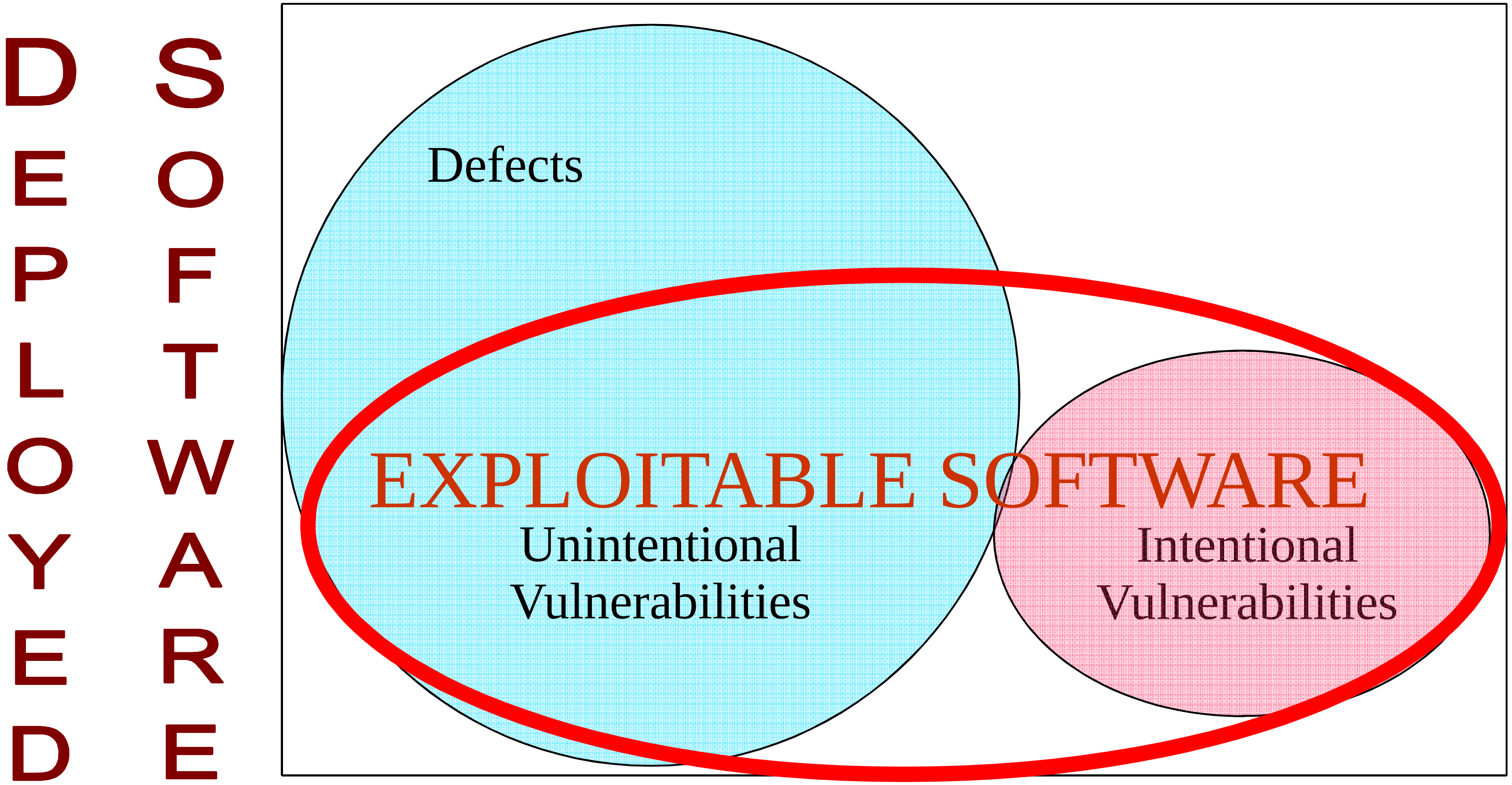
Path Traversal (22)

- Relative Path Traversal (23)
  - Path Traversal: '../filedir' (24)
  - Path Traversal: '/../filedir' (25)
  - <------------8 more here -------------->
  - 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.

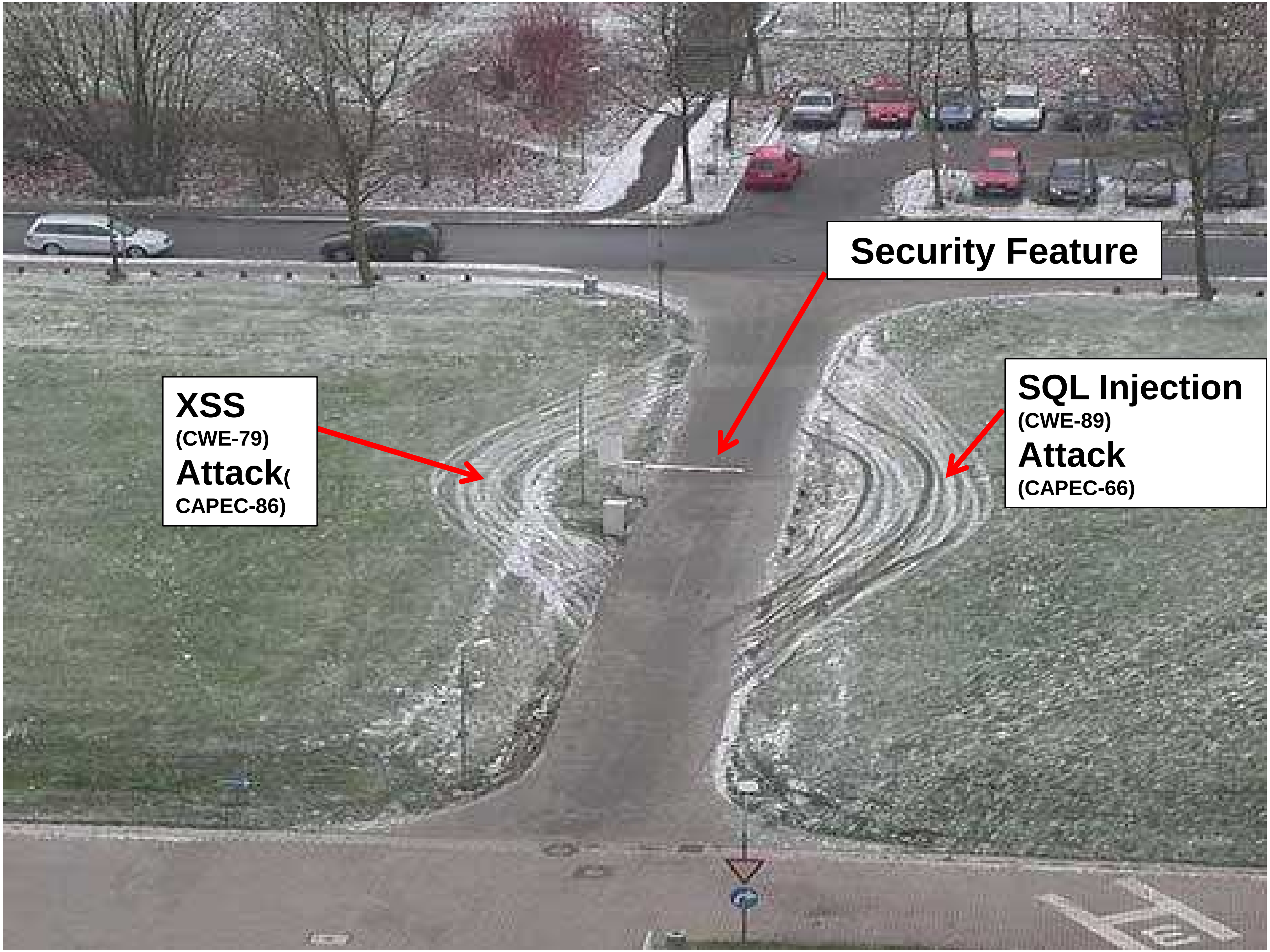


Intentional vulnerabilities are spyware & malicious logic deliberately imbedded (and might not be considered defects but they can make use of the same weakness patterns as unintentional mistakes)

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

But we also need to deal with the people that are out there trying to locate vulnerabilities and weaknesses in our technologies, processes, or practices…

…which could be with defensive and offensive security capabilities.



Security Feature

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

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

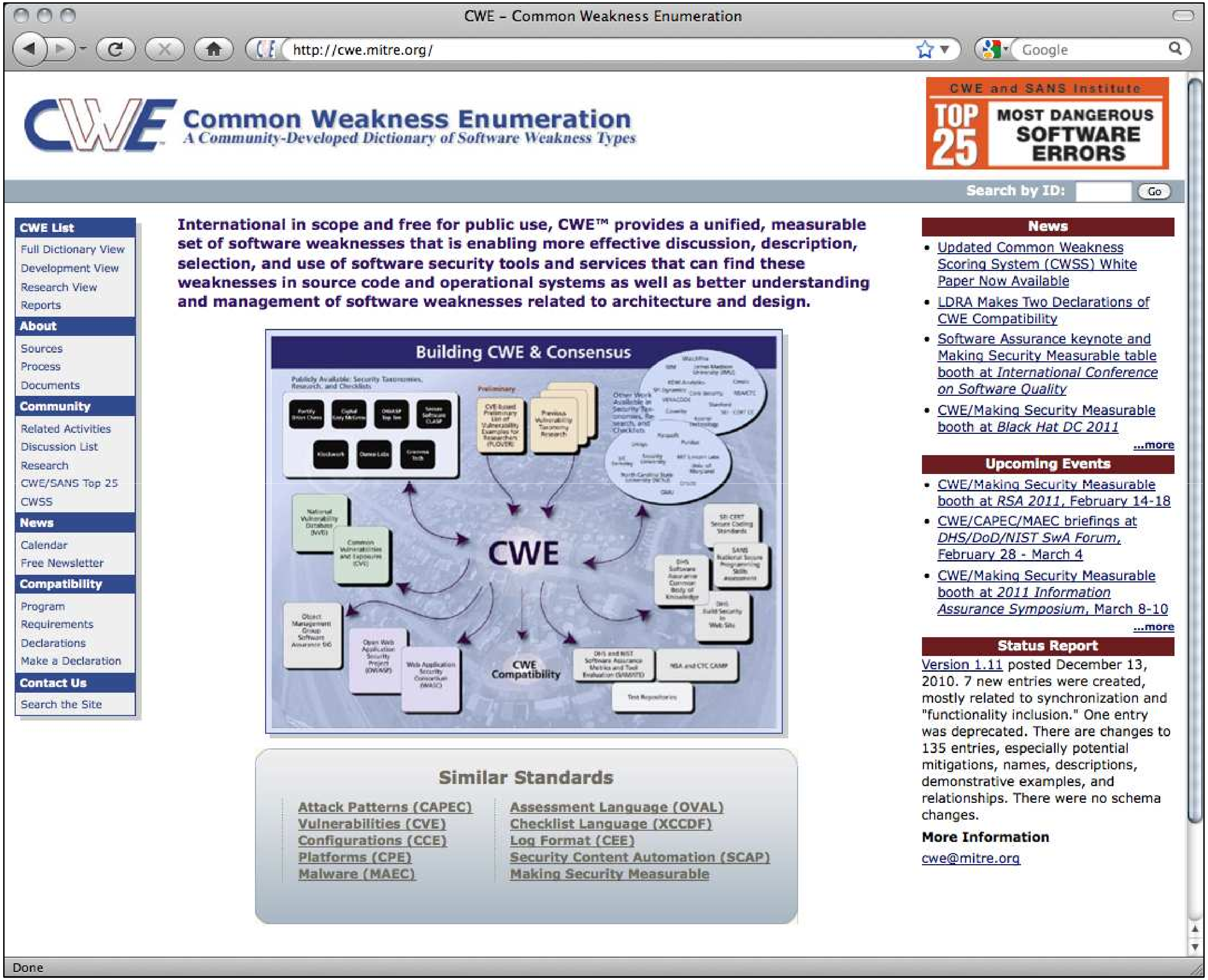
Software [In]security: Cyber Warmongering and Influence Peddling

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.”



Common Weakness Enumeration

A Community-Developed Dictionary of Software Weakness Types

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.

Building CWE & Consensus
Publicly Available: Security Taxonomies, Research, and Checklists
Fortify
Digital
OWASP Top Ten
Secure Software CLASP
Klocwork
Ounce Labs
Preliminary
CVE-based Preliminary List of Vulnerability Examples for Researchers (PLOVER)
Previous Vulnerability Taxonomy Research
Other Work Available in Security Taxonomies, Research, and Checklists
James Madison University (JMU)
North Carolina State University (NCSU)
Univ. of Maryland
National Vulnerability Database (NVD)
Common Vulnerabilities and Exposures (CVE)
CWE
SEI CERT Secure Coding Standards
SANS National Secure Programming Skills Assessment
DHS Software Assurance Common Body of Knowledge
DHS Build Security In Web Site
Object Management Group Software Assurance SIG
Open Web Application Security Project (OWASP)
Web Application Security Consortium (WASC)
CWE Compatibility
DHS and NIST Software Assurance Metrics and Tool Evaluation (SAMATE)
NSA and CTC CAMP
Test Repositories

Similar Standards

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

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

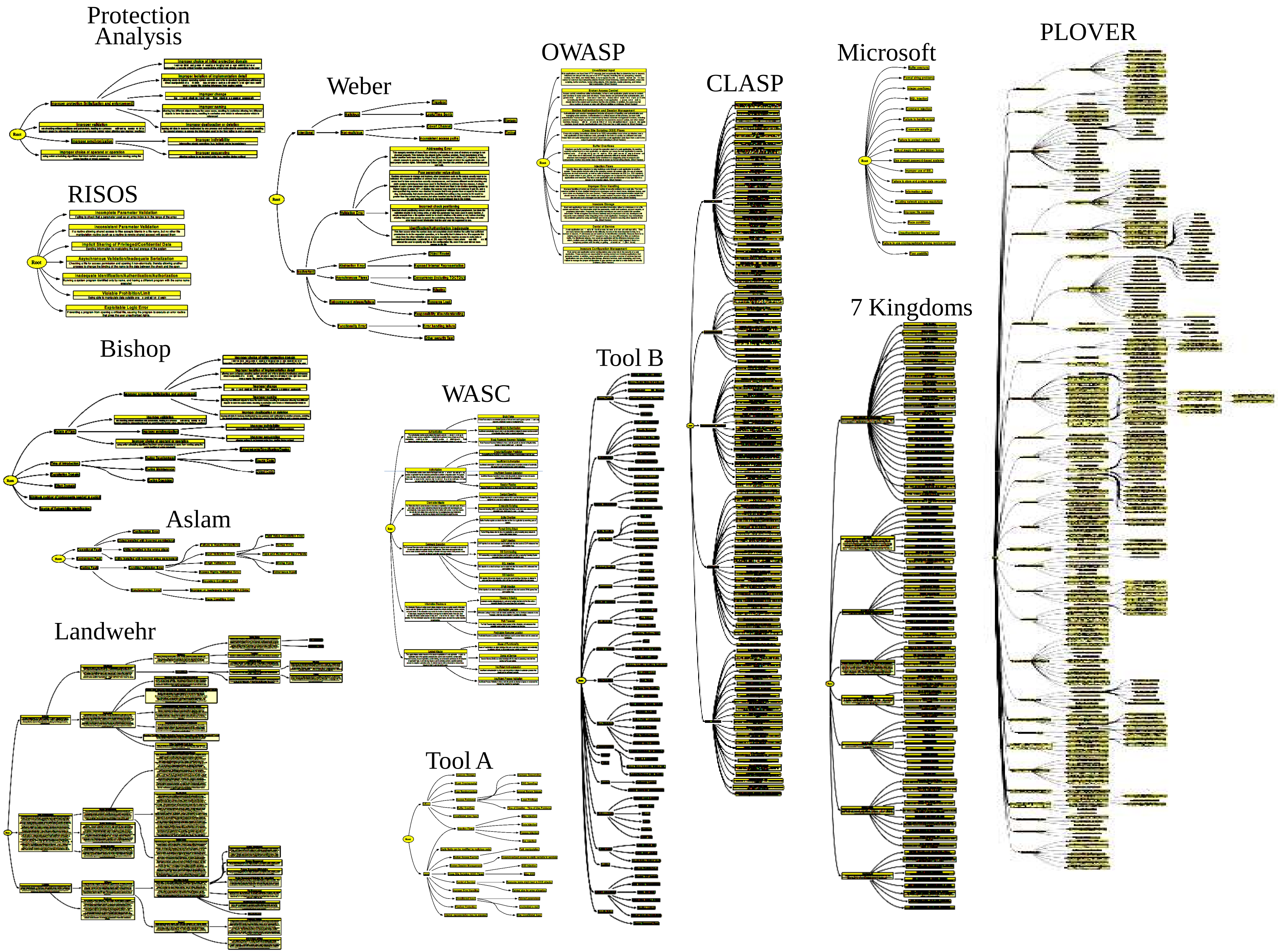
...more

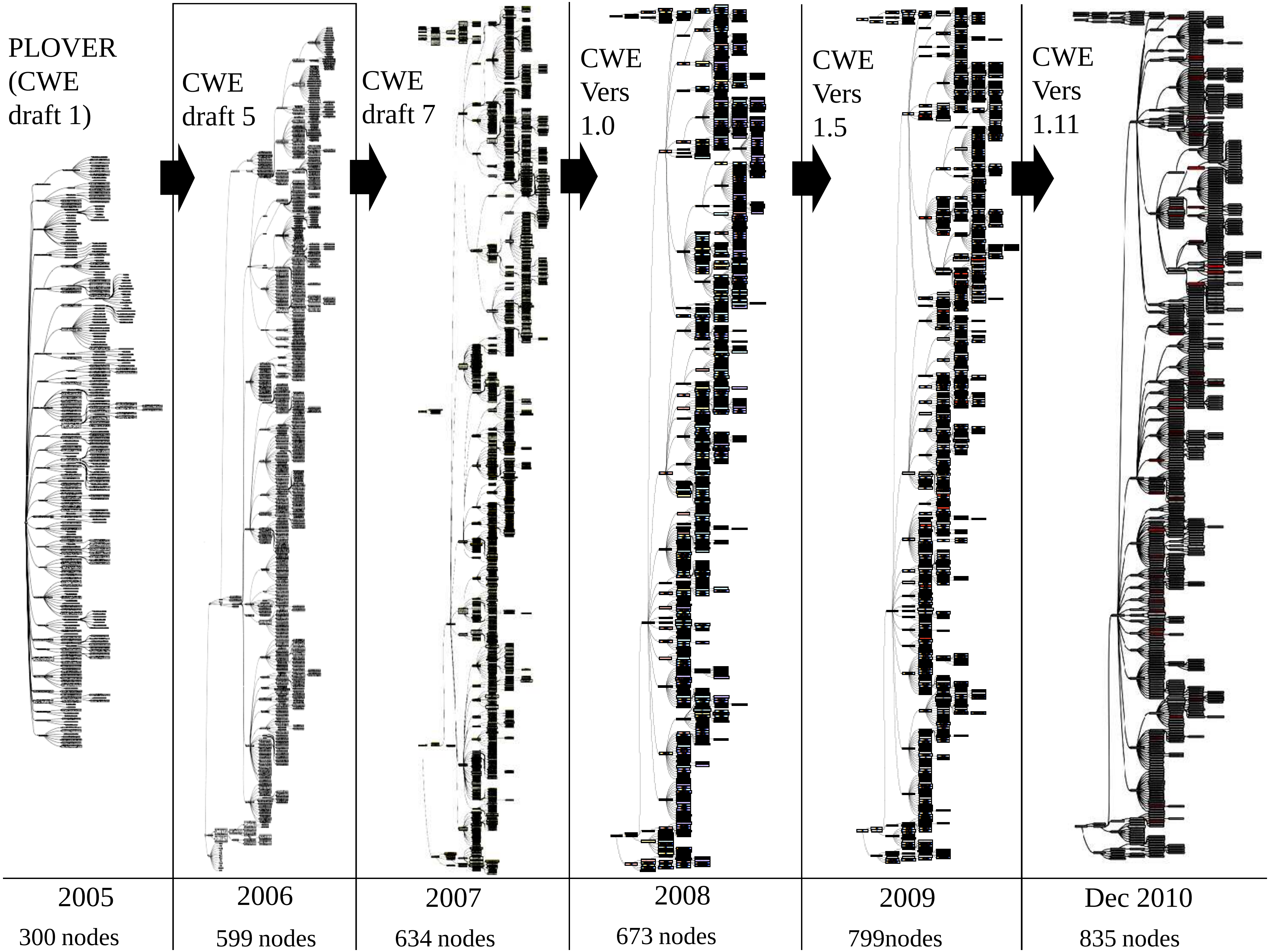
Status Report

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.

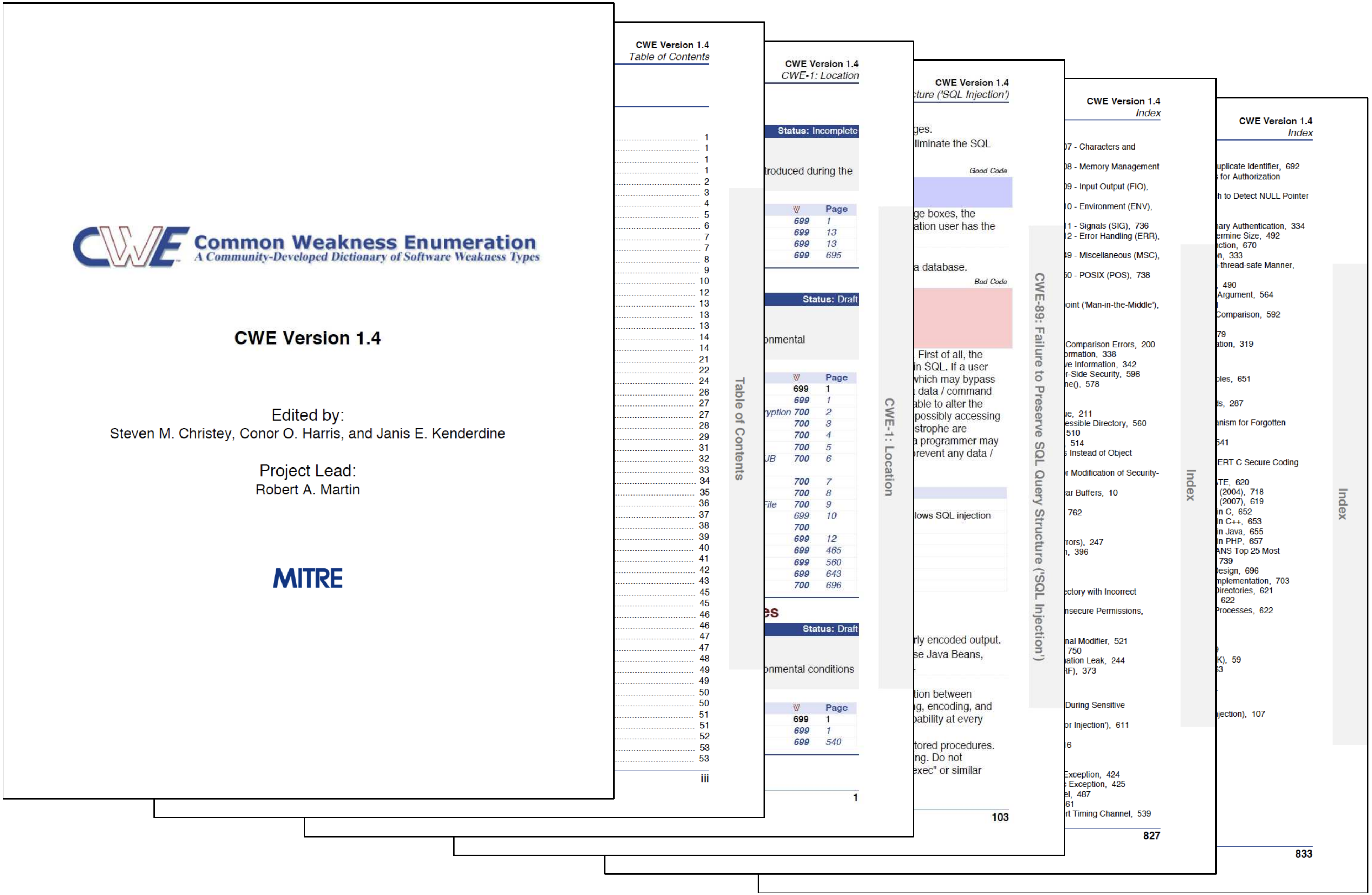
More Information

cwe@mitre.org


Protection Analysis
Root
Improper protection (initialization and enforcement)
Improper choice of initial protection domain
Improper isolation of implementation detail
Improper change
Improper naming
Improper deallocation or deletion
Improper validation
Improper synchronization
Improper indivisibility
Improper sequencing
Improper choice of operand or operation
RISOS
Root
Incomplete Parameter Validation
Failing to check that a parameter used as an array index is in the range of the array
Inconsistent Parameter Validation
If a routine allowing shared access to files accepts blanks in a file name, but no other file manipulation routine (such as a routine to revoke shared access) will accept them
Implicit Sharing of Privileged/Confidential Data
Sending information by modulating the load average of the system
Asynchronous Validation/Inadequate Serialization
Checking a file for access permission and opening it non-atomically, thereby allowing another process to change the binding of the name to the data between the check and the open
Inadequate Identification/Authentication/Authorization
Running a system program identified only by name, and having a different program with the same name executed
Violable Prohibition/Limit
Exploitable Logic Error
Preventing a program from opening a critical file, causing the program to execute an error routine that gives the user unauthorized rights.
Bishop
Root
Nature of Flaw
Time of Introduction
During Development
During Maintenance
During Operation
Requirement/Specification/Design
Source Code
Object Code
Exploitation Domain
Effect Domain
Minimum number of components needed to exploit
Source of Vulnerability Identification
Aslam
Root
Operational Fault
Environment Fault
Coding Fault
Configuration Error
Object installed with incorrect permissions
Utility installed in the wrong place
Utility installed with incorrect setup parameters
Condition Validation Error
Failure to Handle Exceptions
Input Validation Error
Origin Validation Error
Access Rights Validation Error
Boundary Condition Error
Field Value Correlation Error
Syntax Error
Type and Number of Input Fields
Missing Input
Extraneous Input
Synchronization Error
Improper or Inadequate Serialization EError
Race Condition Error
Landwehr
Weber
Root
Intentional
Malicious
Trapdoor
Logic/Time Bomb
Non-malicious
Covert Channel
Storage
Timing
Inconsistent access paths
Inadvertent
Validation Error
Addressing Error
Poor parameter value check
Incorrect check positioning
Identification/Authentication Inadequate
Abstraction Error
Object Reuse
Exposed Internal Representation
Asynchronous Flaws
Concurrency (including TOCTOU)
Aliasing
Subcomponent misuse/failure
Resource Leak
Responsibility Misunderstanding
Functionality Error
Error handling failure
Other security flaw
WASC
Root
Authentication
Brute Force
Insufficient Authentication
Weak Password Recovery Validation
Authorization
Credential/Session Prediction
Insufficient Authorization
Insufficient Session Expiration
Session Fixation
Client-side Attacks
Content Spoofing
Cross-site Scripting
Command Execution
Buffer Overflow
Format String Attack
LDAP Injection
OS Commanding
SQL Injection
SSI Injection
XPath Injection
Information Disclosure
Directory Indexing
Information Leakage
Path Traversal
Predictable Resource Location
Logical Attacks
Abuse of Functionality
Denial of Service
Insufficient Anti-automation
Insufficient Process Validation
Tool A
Root
C/C++
Insecure Storage
Weak Cryptography
Poor Randomization
Access Problems
Improper Sequencing
DNS Spoofing
Ignored Return Values
Least Privilege
Time of Creation - Time of Use Problems
Buffer Overflow
Unvalidated User Input
Injection Flaws
Misc Injection
Data Injection
Process Injection
Sql Injection
Java
Static fields can be modified by malicious code
Path manipulation
Broken Access Control
Unsynchronized access to static variable in servlets
Broken Session Management
XSS Injection
Cross-Site Scripting (XSS) Flaws
Misc XSS
Denial of Service
Resource leaks might lead to DOS attacks
Tainted size for array allocation
Improper Error Handling
Unvalidated Input
Tainted parameters
Unchecked e-mail
Misc Unvalidated Input
Finalizer Protection
Internal representation may be exposed
OWASP
Root
Unvalidated Input
Broken Access Control
Broken Authentication and Session Management
Cross Site Scripting (XSS) Flaws
Buffer Overflows
Injection Flaws
Improper Error Handling
Insecure Storage
Denial of Service
Insecure Configuration Management
Tool B
Root
CLASP
Root
Microsoft
Root
Buffer overruns
Format string problems
Integer overflows
SQL injection
Command injection
Failure to handle errors
Cross-site scripting
Failure to protect network traffic
Use of magic URLs and hidden forms
Use of weak password-based systems
Improper use of SSL
Failure to store and protect data securely
Information leakage
Trusting network address resolution
Improper file accesses
Race conditions
Unauthenticated key exchange
Failure to use cryptographically strong random numbers
Poor usability
7 Kingdoms
Root
PLOVER


PLOVER (CWE draft 1)
CWE draft 5
CWE draft 7
CWE Vers 1.0
CWE Vers 1.5
CWE Vers 1.11
2005
300 nodes
2006
599 nodes
2007
634 nodes
2008
673 nodes
2009
799nodes
Dec 2010
835 nodes

CWE is Meant for People to Use



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

CWE Version 1.4

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

Project Lead:
Robert A. Martin

MITRE

Common Security Errors in Programming

Special thanks to Robert A. Martin of MITRE Corporation.

The SANS Common Security Errors in Programming map illustrates the software weaknesses that are responsible for the majority of the publicly known vulnerabilities discovered in 2008. It is based on the CWE (Common Weakness Enumeration) that provides a unified, measurable set of software weaknesses that will enable more effective discussion and action to find these weaknesses in source code and eliminate them. The CWE was developed by MITRE and sponsored by the Department of Homeland Security. The numbers between parentheses represent the common weakness enumeration IDs for each weakness. Numbers between square brackets are direct children of the weaknesses listed. CWE IDs can be found at the MITRE CWE Website or accessed directly by putting the number (in place of ###) in the following URL:

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

🟧 2009's Top 25 CVE Causes and Important CWEs

Handler Errors

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

User Interface Errors

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

Behavioral Problems

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

Initialization and Cleanup Errors

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

Data Handling

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

Channel and Path Errors

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

Error Handling

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

Pointer Issues

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

Time and State

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

Failure to Fulfill API Contract ('API Abuse')

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

Web Problems

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

Indicator of Poor Code Quality

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

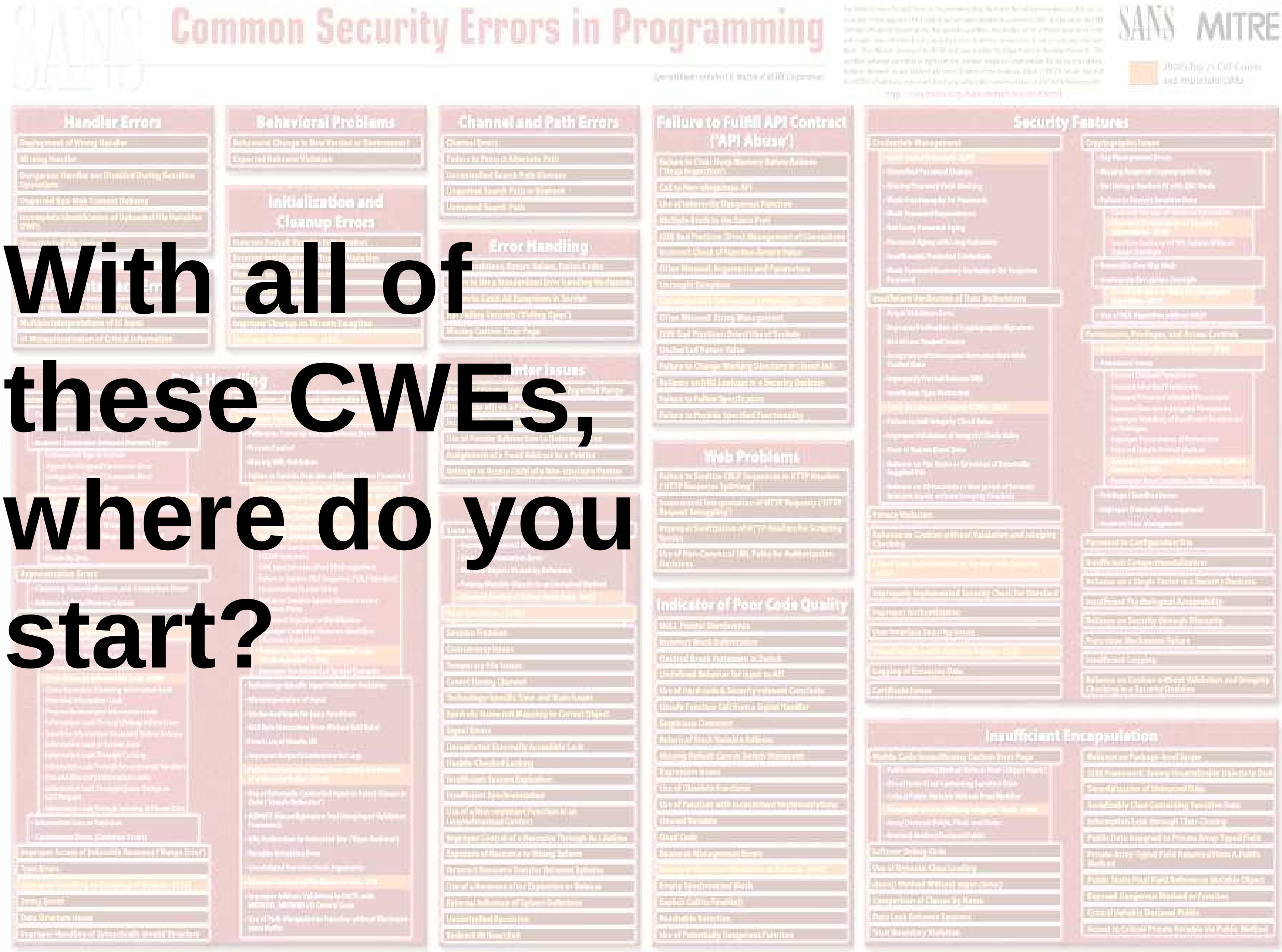
Security Features

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

Insufficient Encapsulation

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

With all of these CWEs, where do you start?



Common Security Errors in Programming

SANS MITRE

2009's Top 25 CWE Errors and important CWEs

Handler Errors

Behavioral Problems

Channel and Path Errors

Failure to Fulfill API Contract ('API Abuse')

Security Features

Initialization and Cleanup Errors

Error Handling

Data Handling

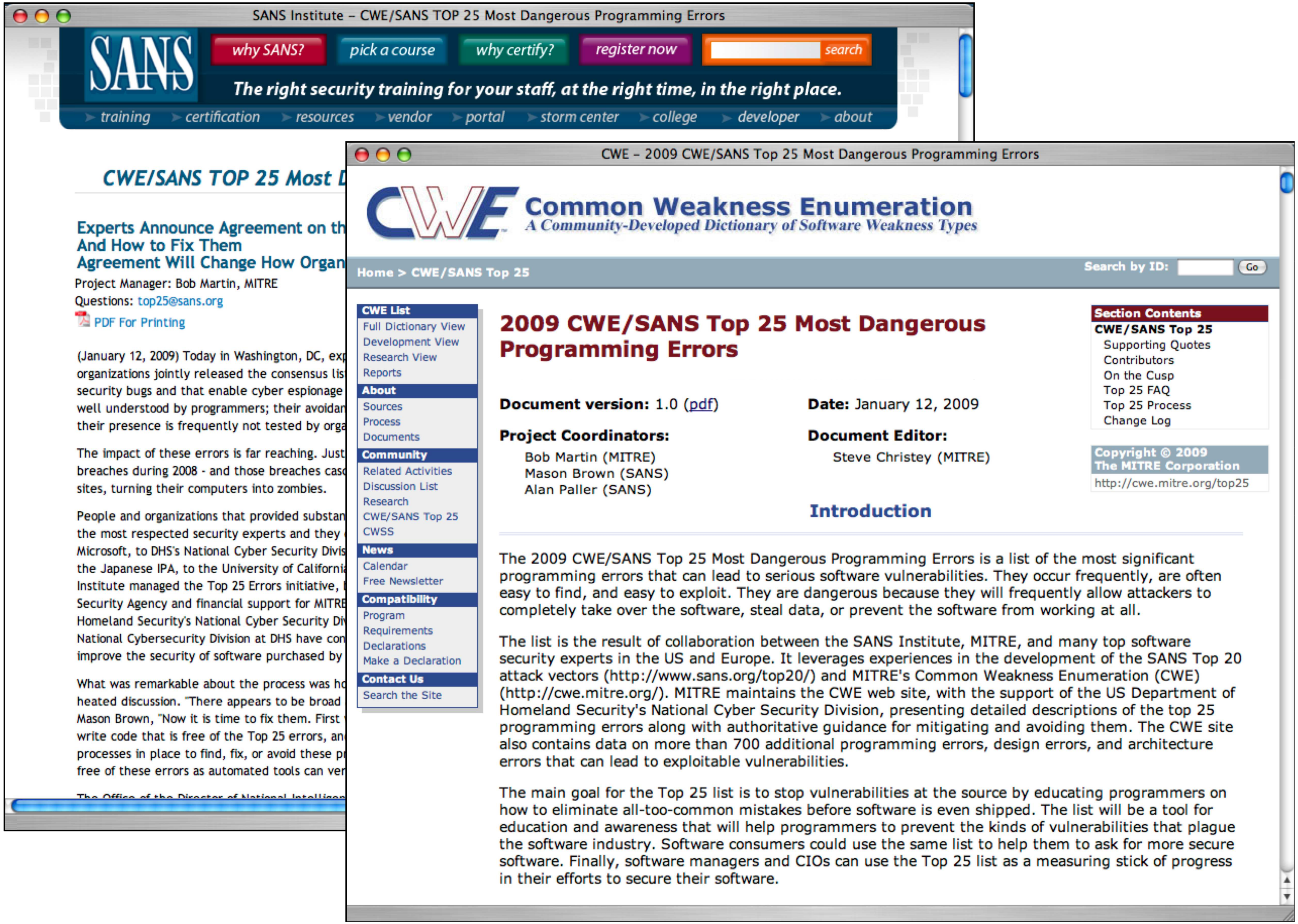
Web Problems

Indicator of Poor Code Quality

Insufficient Encapsulation

2009 SANS/CWE Top 25 Programming Errors

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



SANS Institute - CWE/SANS TOP 25 Most Dangerous Programming Errors
SANS
why SANS?
pick a course
why certify?
register now
search
The right security training for your staff, at the right time, in the right place.
training
certification
resources
vendor
portal
storm center
college
developer
about

CWE/SANS TOP 25 Most D

Experts Announce Agreement on th
And How to Fix Them
Agreement Will Change How Organ

Project Manager: Bob Martin, MITRE
Questions: top25@sans.org
PDF For Printing

(January 12, 2009) Today in Washington, DC, exp
organizations jointly released the consensus lis
security bugs and that enable cyber espionage
well understood by programmers; their avoidan
their presence is frequently not tested by orga

The impact of these errors is far reaching. Just
breaches during 2008 - and those breaches casc
sites, turning their computers into zombies.

People and organizations that provided substan
the most respected security experts and they
Microsoft, to DHS's National Cyber Security Divis
the Japanese IPA, to the University of Californi
Institute managed the Top 25 Errors initiative,
Security Agency and financial support for MITRE
Homeland Security's National Cyber Security Di
National Cybersecurity Division at DHS have con
improve the security of software purchased by

What was remarkable about the process was ho
heated discussion. "There appears to be broad
Mason Brown, "Now it is time to fix them. First
write code that is free of the Top 25 errors, an
processes in place to find, fix, or avoid these p
free of these errors as automated tools can ver

2009 CWE/SANS Top 25 Most Dangerous Programming Errors

Document version: 1.0 (pdf)

Date: January 12, 2009

Project Coordinators:

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

Document Editor:

Steve Christey (MITRE)

Introduction

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.

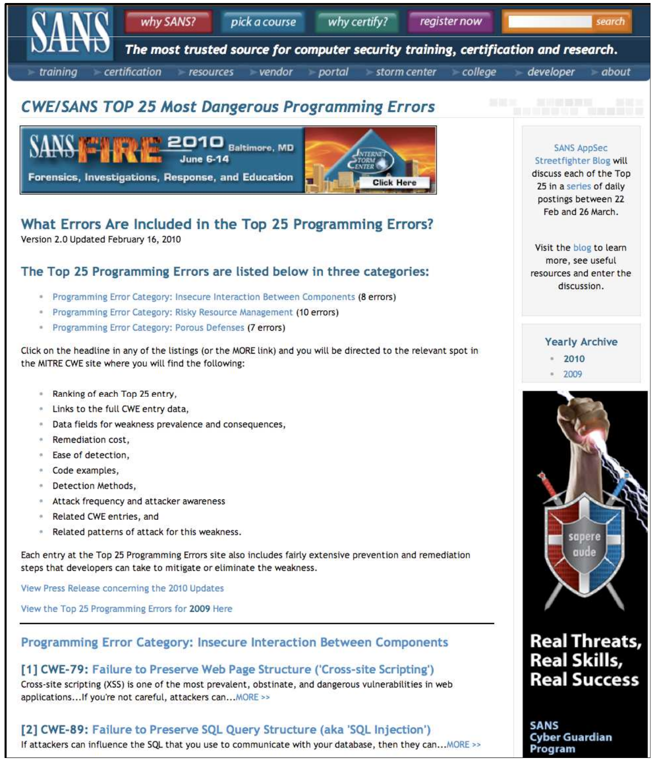
The list is the result of collaboration between the SANS Institute, MITRE, and many top software security experts in the US and Europe. It leverages experiences in the development of the SANS Top 20 attack vectors (http://www.sans.org/top20/) and MITRE's Common Weakness Enumeration (CWE) (http://cwe.mitre.org/). MITRE maintains the CWE web site, with the support of the US Department of Homeland Security's National Cyber Security Division, presenting detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigating and avoiding them. The CWE site also contains data on more than 700 additional programming errors, design errors, and architecture errors that can lead to exploitable vulnerabilities.

The main goal for the Top 25 list is to stop vulnerabilities at the source by educating programmers on how to eliminate all-too-common mistakes before software is even shipped. The list will be a tool for education and awareness that will help programmers to prevent the kinds of vulnerabilities that plague the software industry. Software consumers could use the same list to help them to ask for more secure software. Finally, software managers and CIOs can use the Top 25 list as a measuring stick of progress in their efforts to secure their software.

20010 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, WhiteHat
  - – Security Groups: OWASP, WASC



CWE/SANS TOP 25 Most Dangerous Programming Errors

SANS FIRE 2010 Baltimore, MD
June 6-14
Forensics, Investigations, Response, and Education
Internet Storm Center
Click Here

What Errors Are Included in the Top 25 Programming Errors?

Version 2.0 Updated February 16, 2010

The Top 25 Programming Errors are listed below in three categories:

- Programming Error Category: Insecure Interaction Between Components (8 errors)
- Programming Error Category: Risky Resource Management (10 errors)
- Programming Error Category: Porous Defenses (7 errors)

Click on the headline in any of the listings (or the MORE link) and you will be directed to the relevant spot in the MITRE CWE site where you will find the following:

- Ranking of each Top 25 entry,
- Links to the full CWE entry data,
- Data fields for weakness prevalence and consequences,
- Remediation cost,
- Ease of detection,
- Code examples,
- Detection Methods,
- Attack frequency and attacker awareness
- Related CWE entries, and
- Related patterns of attack for this weakness.

Each entry at the Top 25 Programming Errors site also includes fairly extensive prevention and remediation steps that developers can take to mitigate or eliminate the weakness.

View Press Release concerning the 2010 Updates

View the Top 25 Programming Errors for **2009** Here

Programming Error Category: Insecure Interaction Between Components

[1] CWE-79: Failure to Preserve Web Page Structure ('Cross-site Scripting')

Cross-site scripting (XSS) is one of the most prevalent, obstinate, and dangerous vulnerabilities in web applications...If you're not careful, attackers can...MORE >>

[2] CWE-89: Failure to Preserve SQL Query Structure (aka 'SQL Injection')

If attackers can influence the SQL that you use to communicate with your database, then they can...MORE >>

SANS AppSec Streetfighter Blog will discuss each of the Top 25 in a series of daily postings between 22 Feb and 26 March.

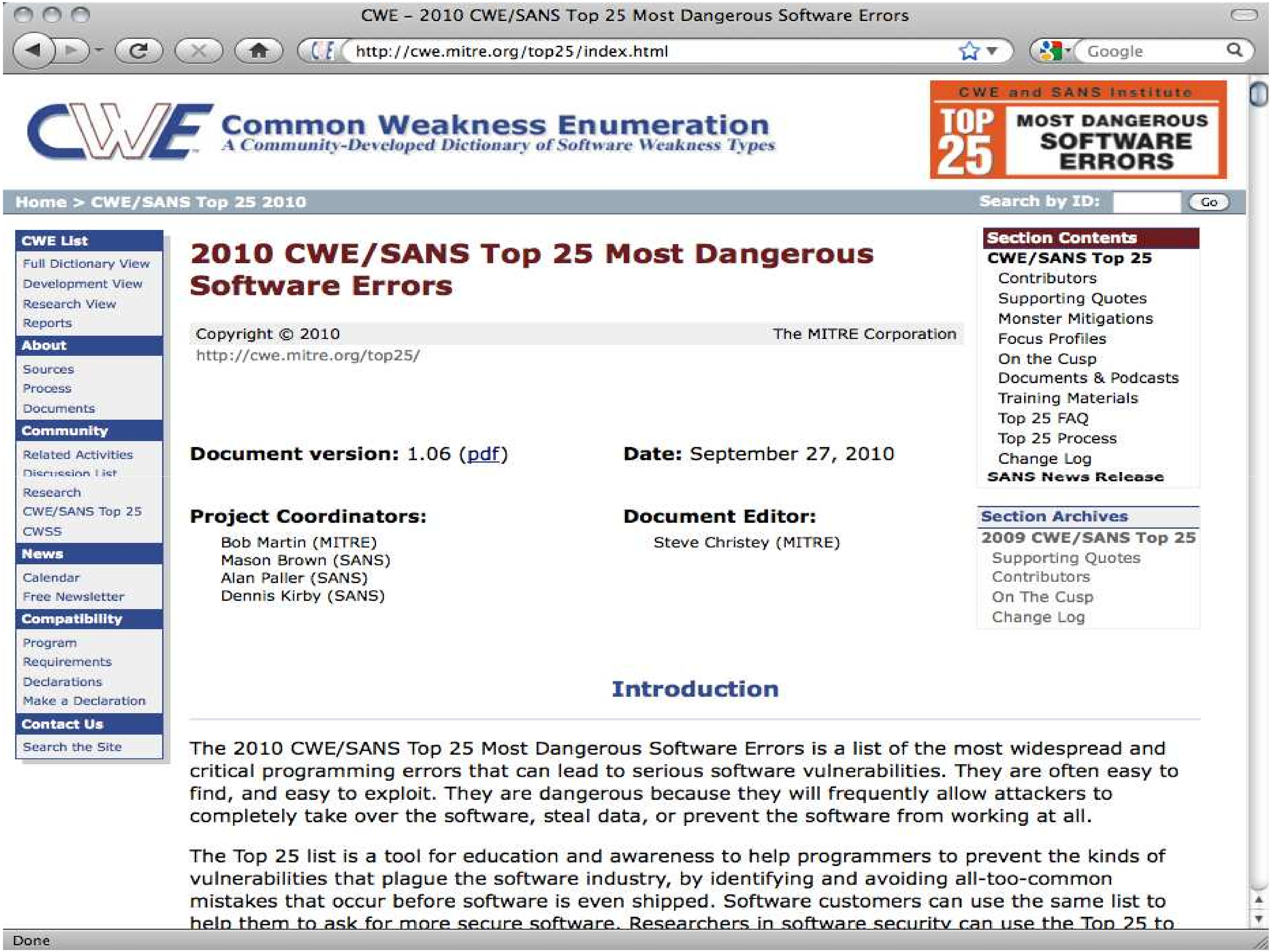
Visit the blog to learn more, see useful resources and enter the discussion.

Yearly Archive
- 2010
- 2009

sapere aude
Real Threats, Real Skills, Real Success
SANS Cyber Guardian Program

Top 25 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



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)

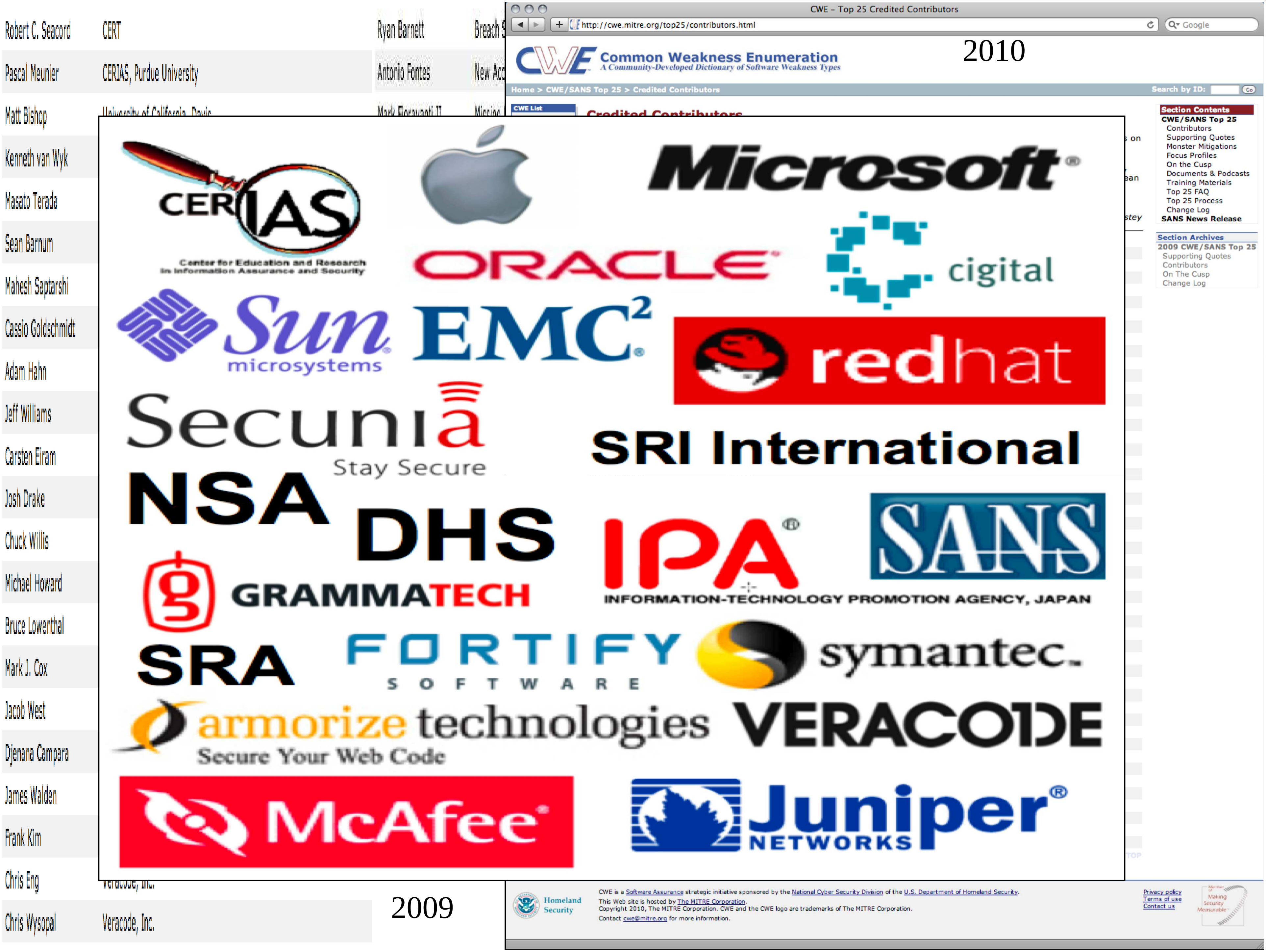
Document Editor:

Steve Christey (MITRE)

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 S
Pascal Meunier	CERIAS, Purdue University	Antonio Fontes	New Acc
Matt Bishop	University of California, Davis	Mark Fiorvanti II	Missing
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.		

2009

CWE – Top 25 Credited Contributors

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

2010

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

Home > CWE/SANS Top 25 > Credited Contributors

Search by ID: Go

CWE List

Credited Contributors

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**

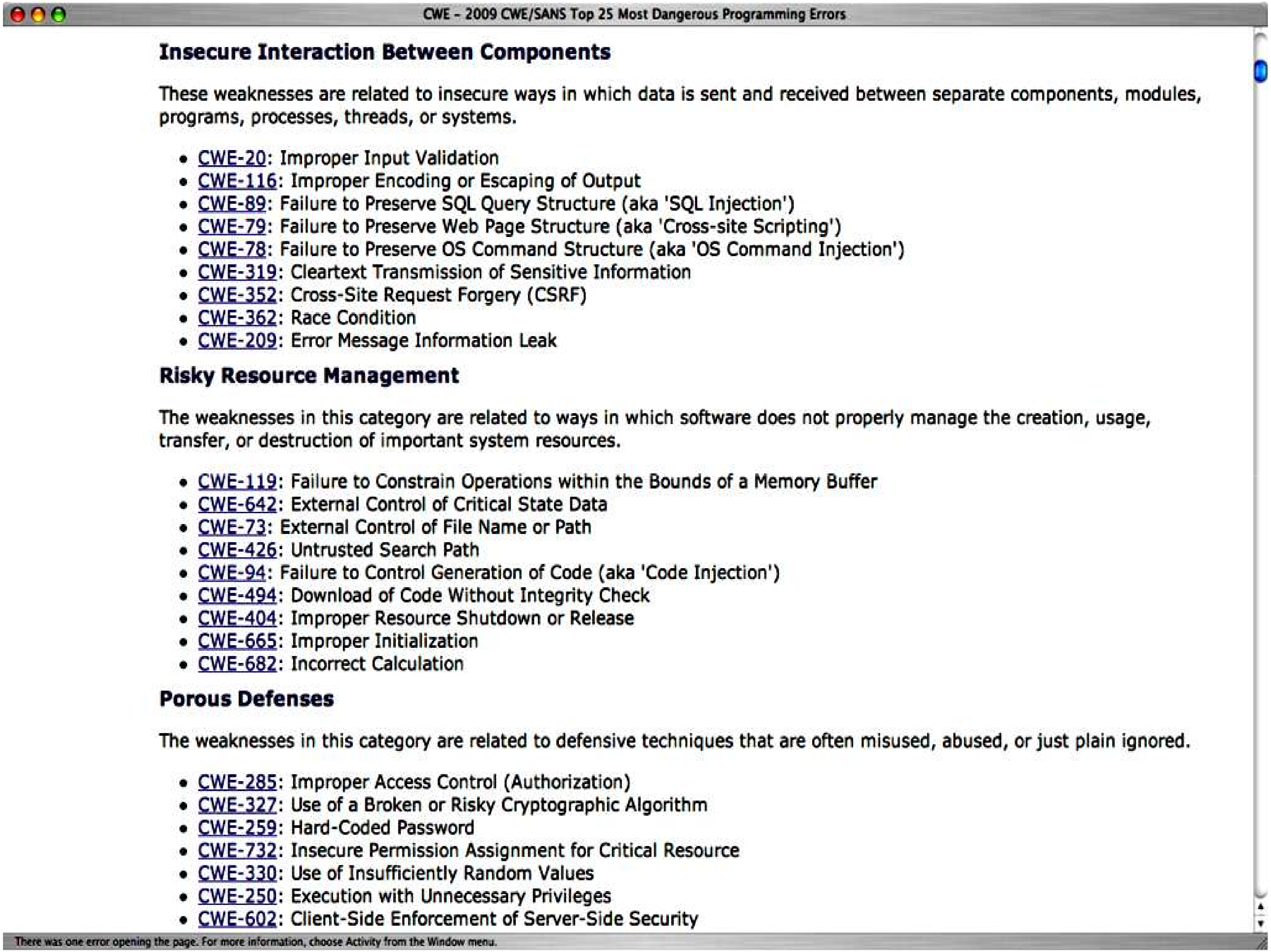
Section Archives
- **2009 CWE/SANS Top 25**
  - Supporting Quotes
  - Contributors
  - On The Cusp
  - Change Log

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

Google
CERIAS
Center for Education and Research in Information Assurance and Security
Microsoft
ORACLE
cigital
Sun microsystems
EMC2
redhat
Secunia Stay Secure
SRI International
NSA
DHS
IPA
INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN
SANS
GRAMMATECH
SRA
FORTIFY SOFTWARE
symantec.
armorize technologies Secure Your Web Code
VERACODE
McAfee
Juniper NETWORKS
Making Security Measurable
TOP


CWE - 2009 CWE/SANS Top 25 Most Dangerous Programming Errors

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.

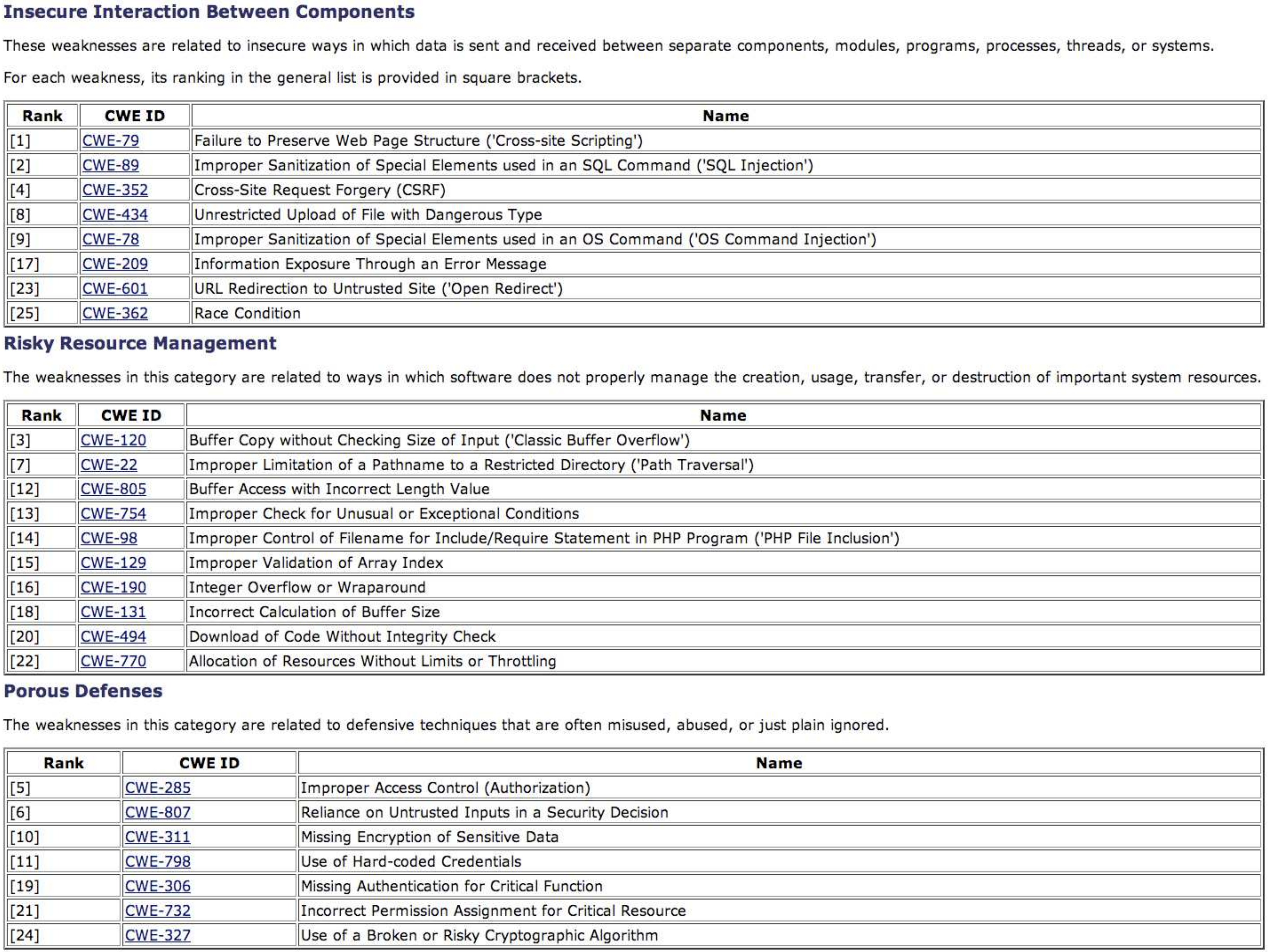
- CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer
- CWE-642: External Control of Critical State Data
- CWE-73: External Control of File Name or Path
- CWE-426: Untrusted Search Path
- CWE-94: Failure to Control Generation of Code (aka 'Code Injection')
- CWE-494: Download 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

There was one error opening the page. For more information, choose Activity from the Window menu.



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

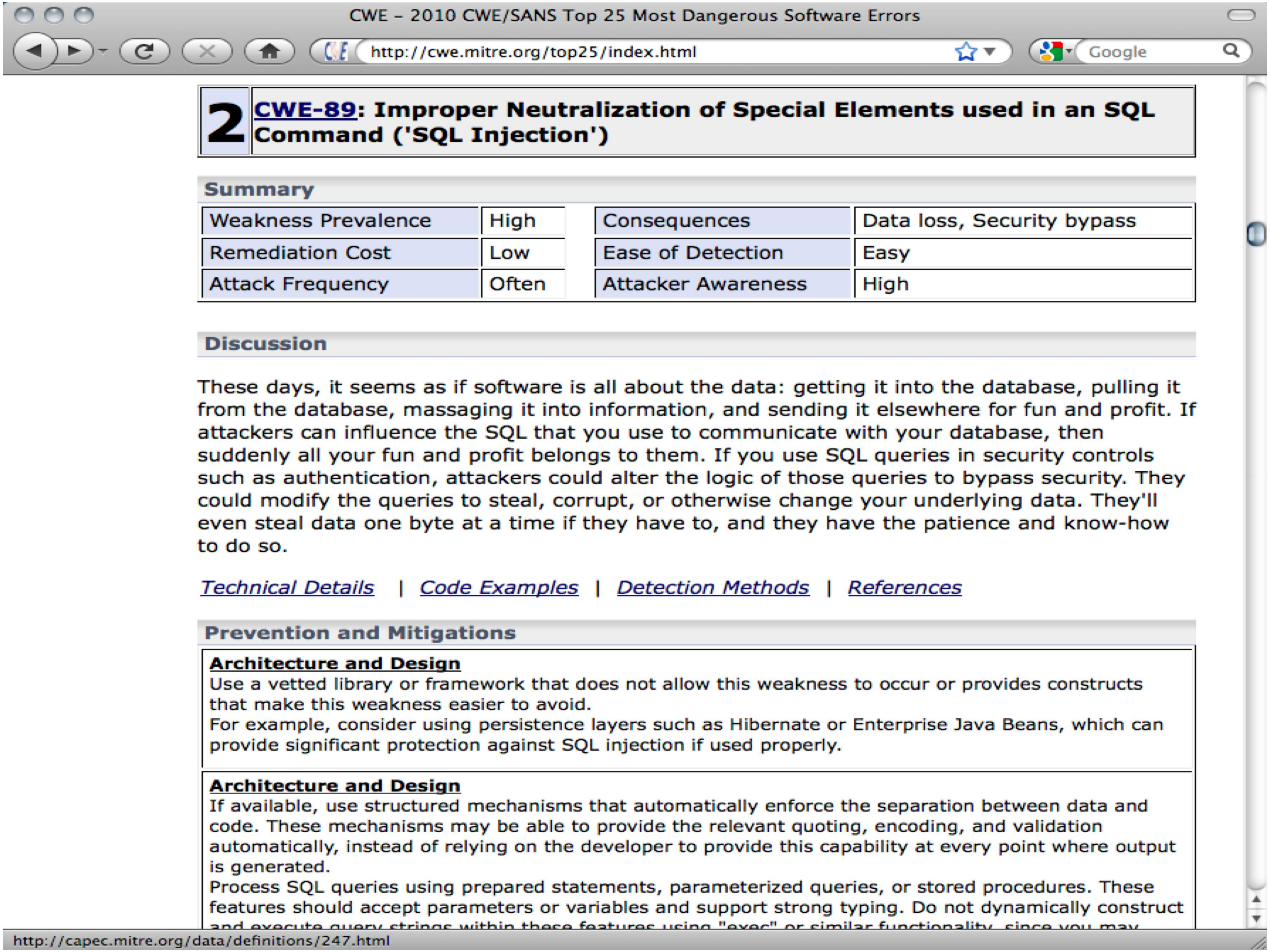
The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

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


CWE – 2010 CWE/SANS Top 25 Most Dangerous Software Errors
http://cwe.mitre.org/top25/index.html
Google
http://capec.mitre.org/data/definitions/247.html

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

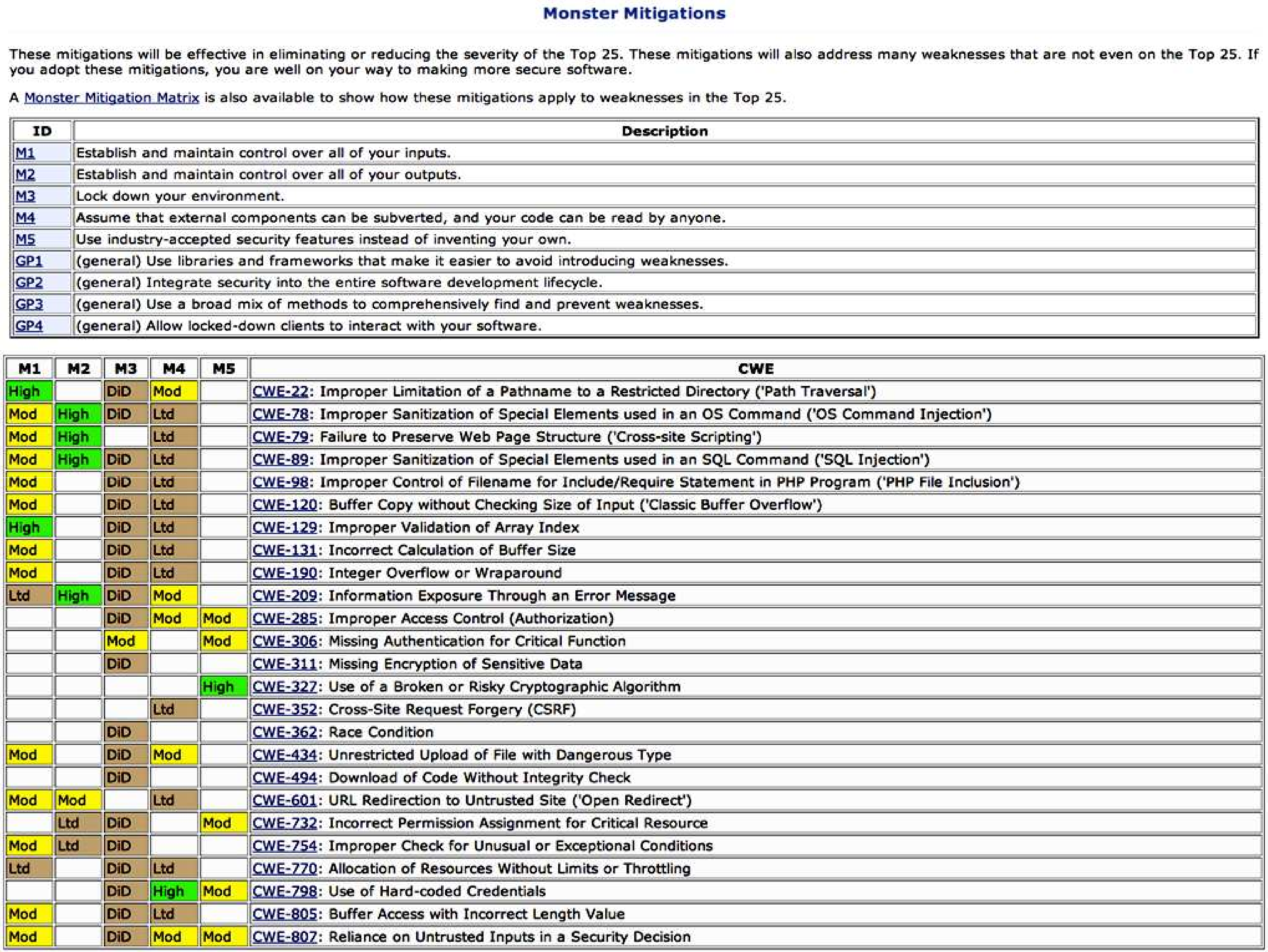
These days, it seems as if software is all about the data: getting it into the database, pulling it from the database, massaging it into information, and sending it elsewhere for fun and profit. If attackers can influence the SQL that you use to communicate with your database, then suddenly all your 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



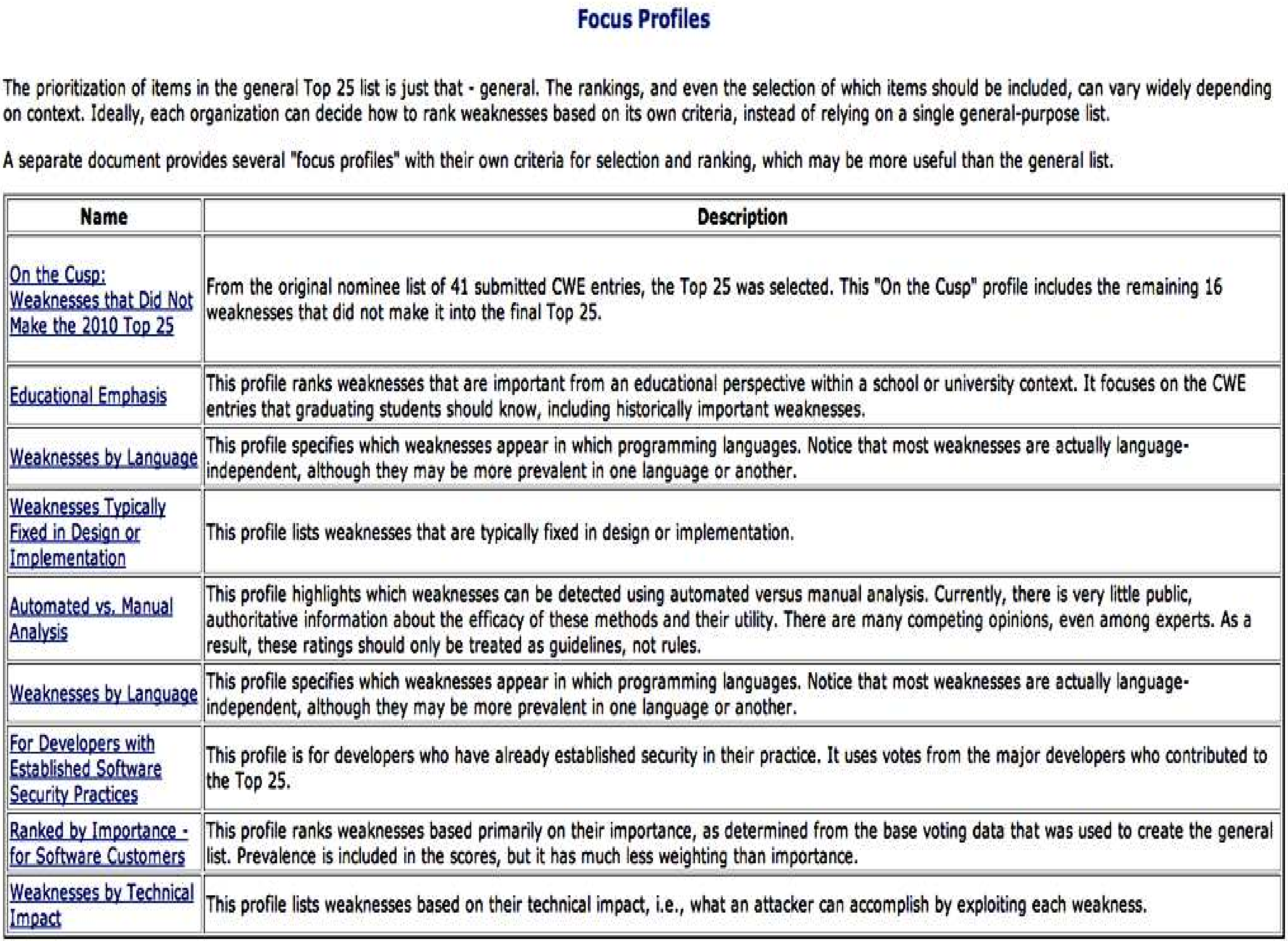
Monster Mitigations

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.

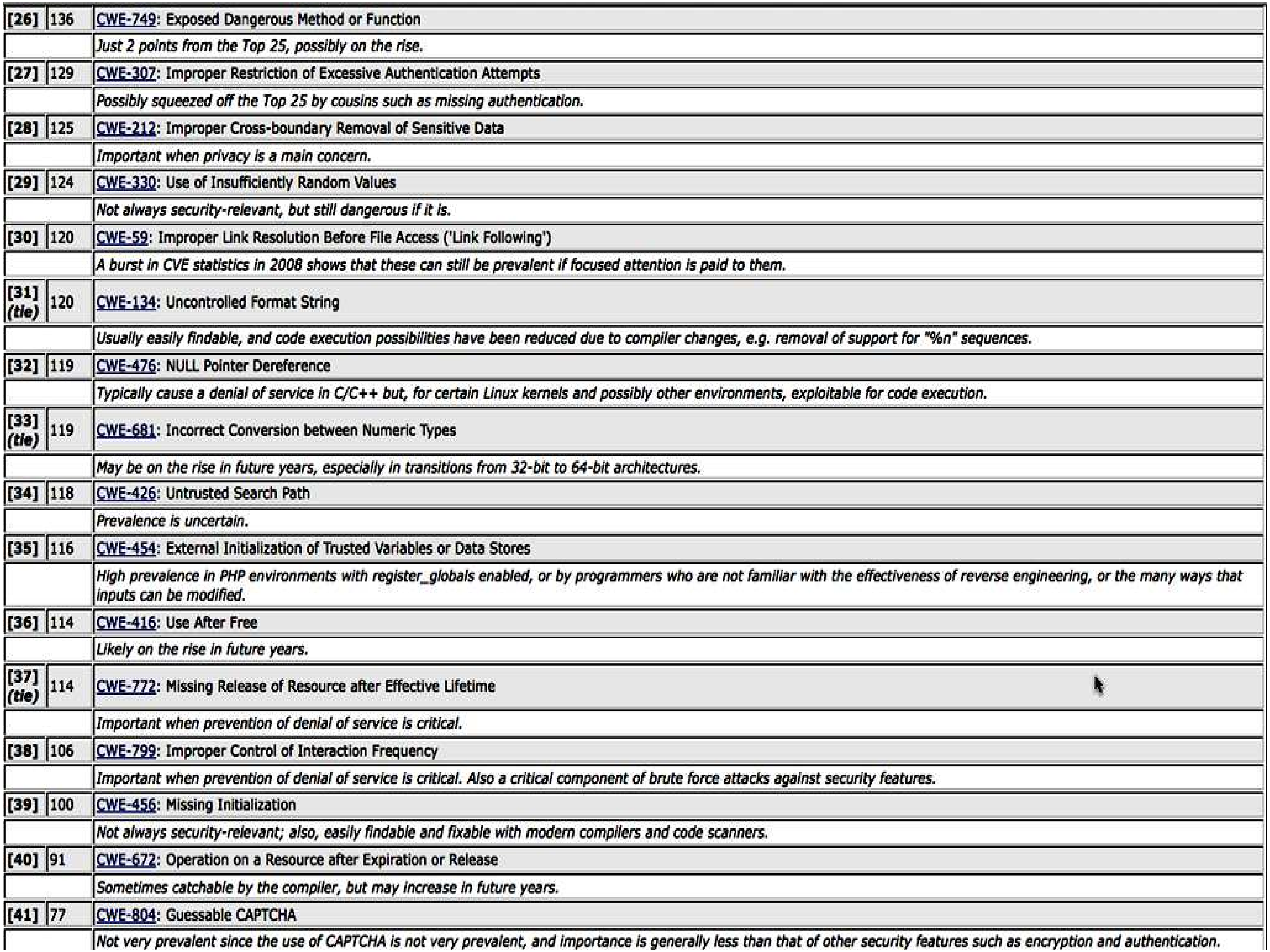
A separate document provides several "focus profiles" with their own criteria for selection and ranking, which may be more useful than the general list.

Name	Description
On the Cusp: Weaknesses that Did Not Make the 2010 Top 25	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.
Educational Emphasis	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.
Weaknesses by Language	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.
Weaknesses Typically Fixed in Design or Implementation	This profile lists weaknesses that are typically fixed in design or implementation.
Automated vs. Manual Analysis	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.
Weaknesses by Language	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.
For Developers with Established Software Security Practices	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.
Ranked by Importance - for Software Customers	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.
Weaknesses by Technical Impact	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



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

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.

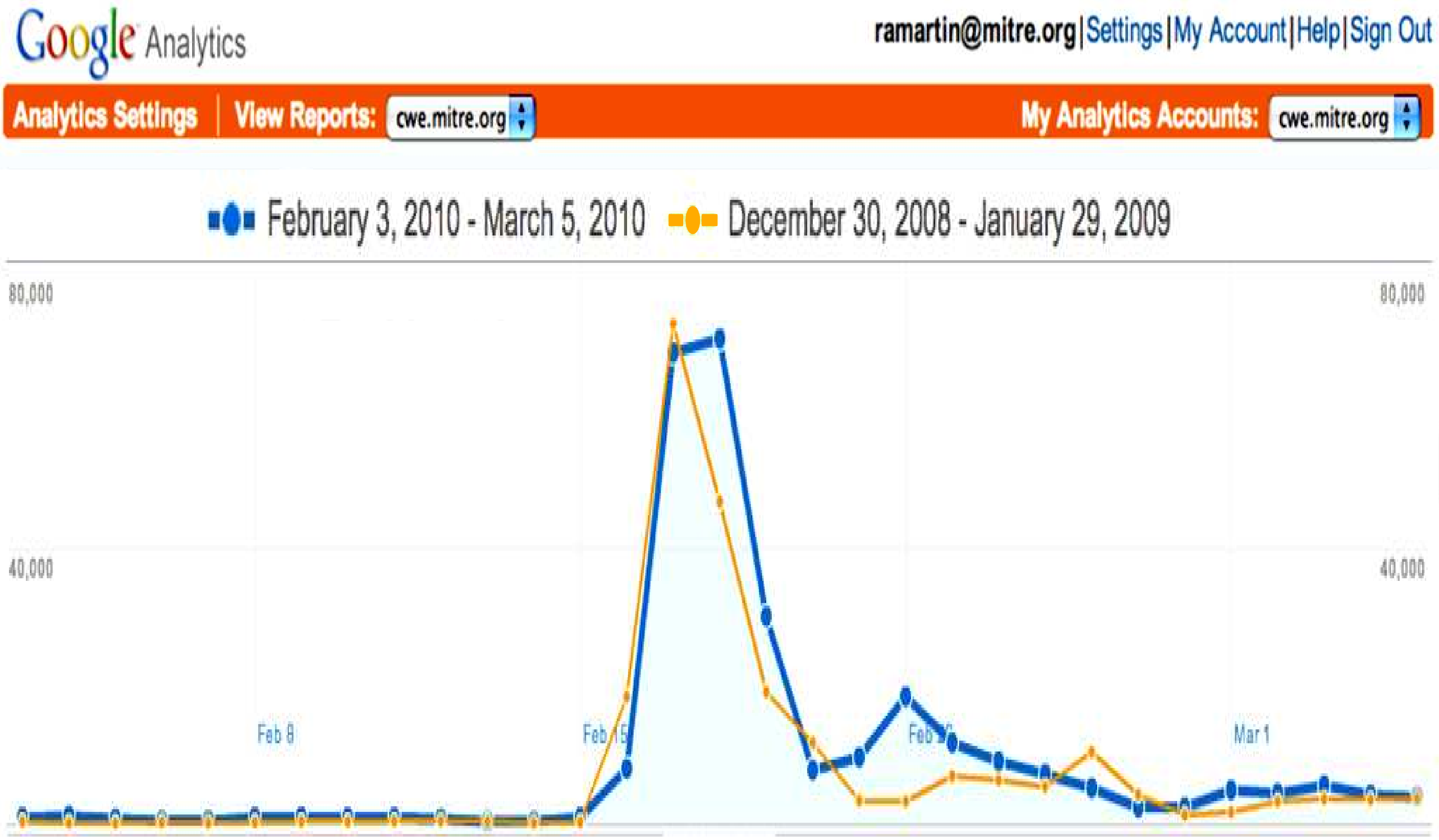
Finally, it was a conscious decision that if there was enough prevalence and severity, design-related weaknesses would be included. These are often thought of as being more abstract than weaknesses that arise during implementation.

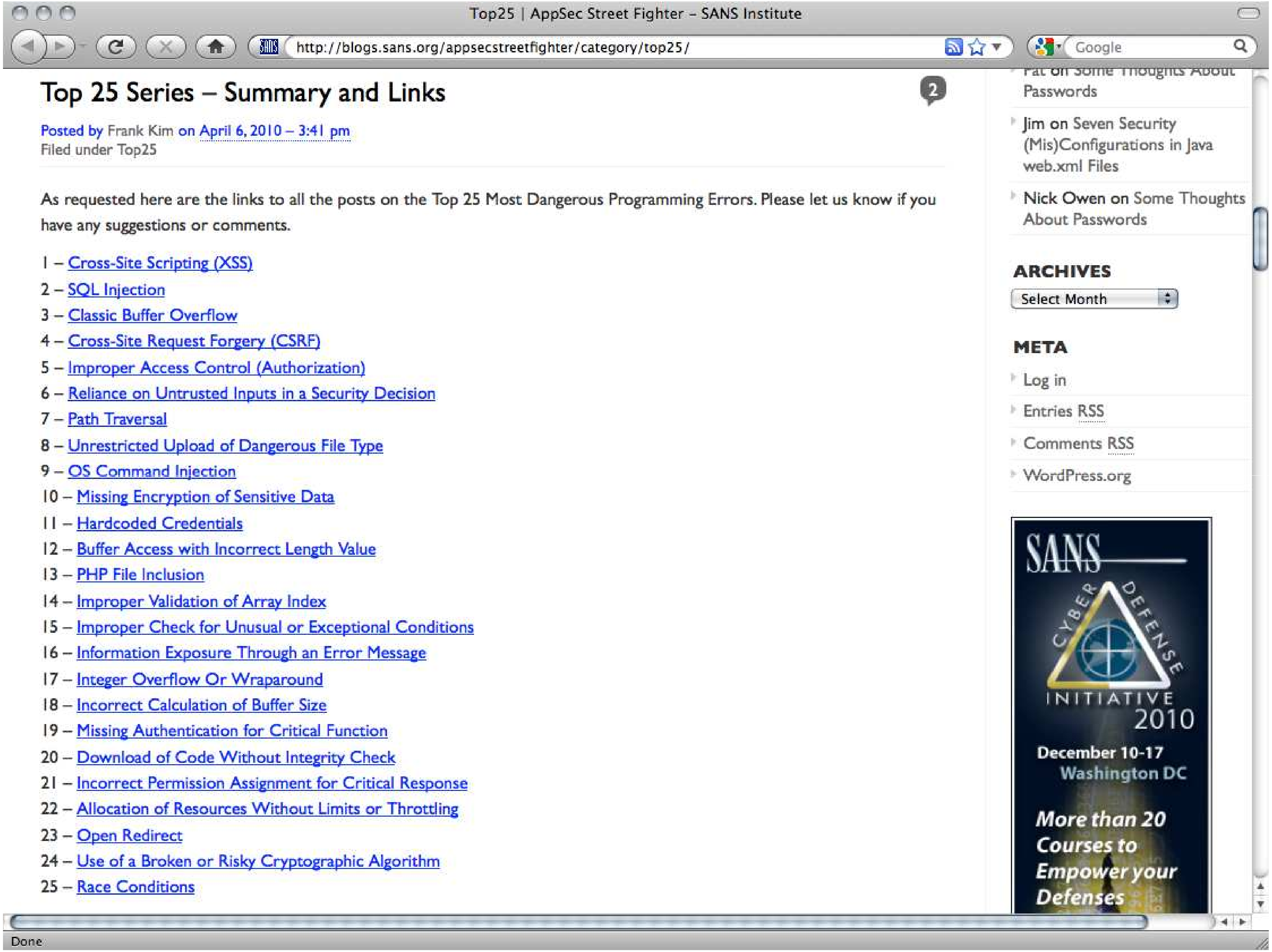
The Top 25 list tries to strike a delicate balance between usability and relevance, and we believe that it does so, even with this apparent imperfection.

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: cwe.mitre.org
My Analytics Accounts: cwe.mitre.org
February 3, 2010 - March 5, 2010
December 30, 2008 - January 29, 2009
80,000
40,000
80,000
40,000
Feb 8
Feb 15
Feb 22
Mar 1


Top25 | AppSec Street Fighter - SANS Institute
http://blogs.sans.org/appsecstreetfighter/category/top25/
Google
Done

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

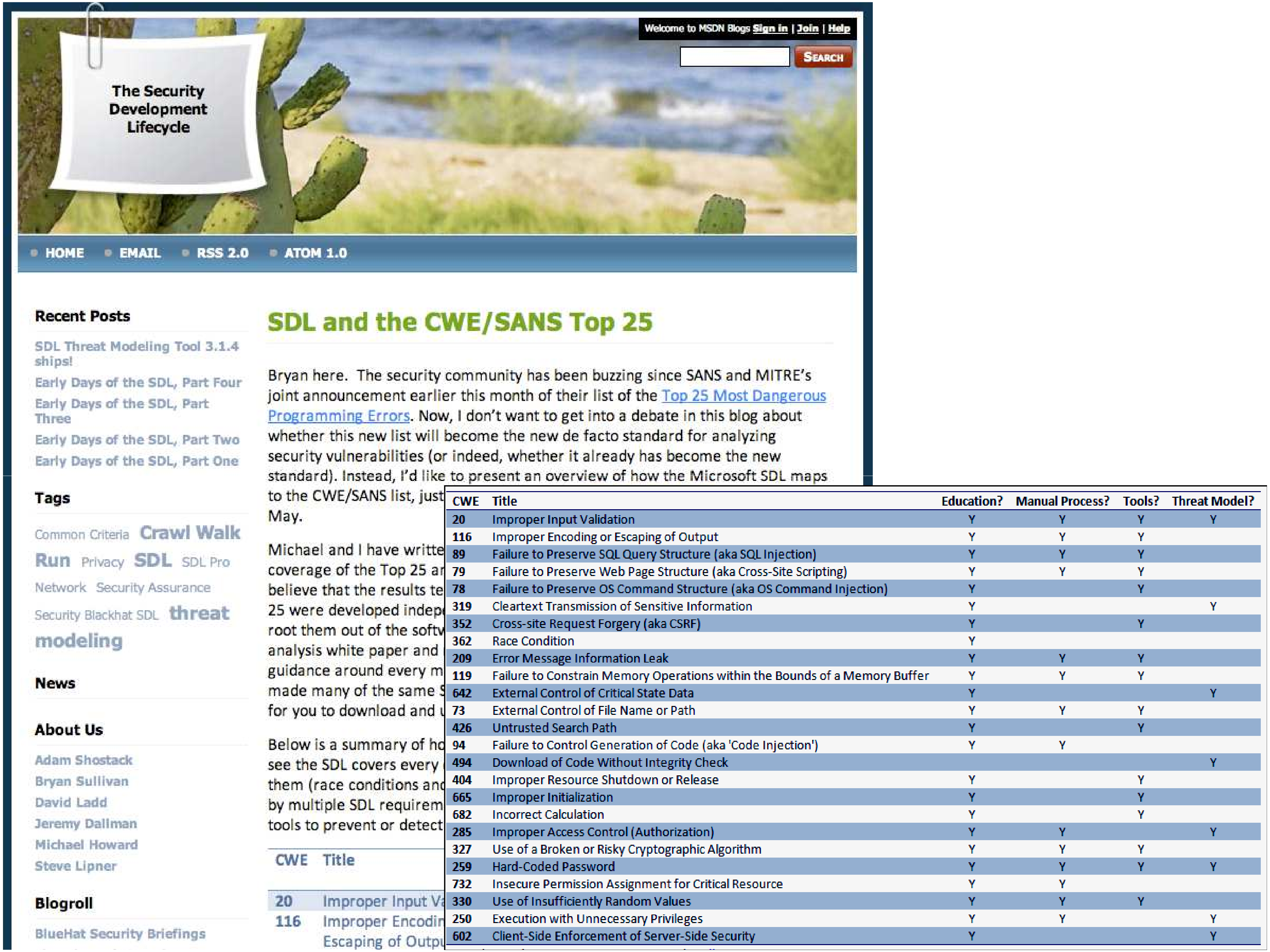
- Pat on Some Thoughts About Passwords
- Jim on Seven Security (Mis)Configurations in Java web.xml Files
- Nick Owen on Some Thoughts About Passwords

ARCHIVES

Select Month

META

- Log in
- Entries RSS
- Comments RSS
- WordPress.org



Welcome to MSDN Blogs Sign in | Join | Help
Search
The Security Development Lifecycle
HOME
EMAIL
RSS 2.0
ATOM 1.0

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 writte
coverage of the Top 25 ar
believe that the results te
25 were developed indep
root them out of the softw
analysis white paper and
guidance around every m
made many of the same S
for you to download and u

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

CWE	Title
20	Improper Input Va
116	Improper Encodin Escaping of Outpu

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…

(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 Samy 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.

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-209: Error Message Information Leak

Error information is critical to debugging failed operations, but you must understand who can read that data. In general, you should restrict detailed error messages to trusted users. Remote and anonymous users should see generic messages with the detailed data logged to an audit log.

Basic Training

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

Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

Michael Howard
Microsoft

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 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-

68
COPUBLISHED BY THE IEEE COMPUTER AND RELIABILITY SOCIETIES
Reading Over Your Shoulder • Dealing with the Smart Grid
IEEE SECURITY & PRIVACY
May/June 2009
Volume 7, Number 3
SECURING ONLINE GAMES

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 0.3) 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.

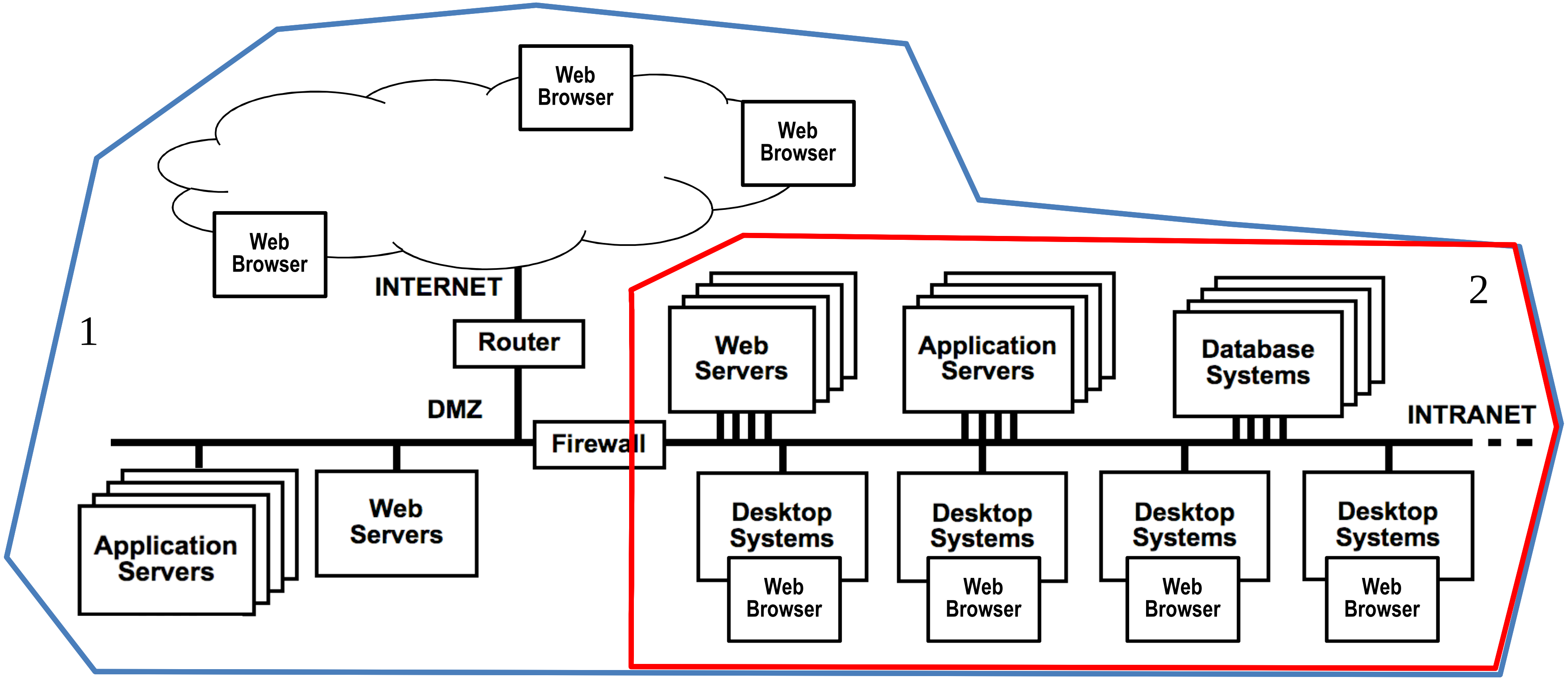
Common Weakness Scoring System (CWSS)

Archetypes:

- Web Browser User Interface
- Web Servers
- Application Servers
- Database Systems
- Desktop Systems
- SSL

Vignettes:

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



Business Value Context (BVC)

- Identifies critical assets and security concerns
- Links Technical Impacts (derived from CWE weaknesses) with business implications
- More fine-grained model than the CIA Triad

CWE Technical Impacts

1. **Modify memory**
2. **Read memory**
3. **Modify files or directories**
4. **Read files or directories**
5. **Modify application data**
6. **Read application data**
7. **DoS: crash / exit / restart**
8. **DoS: amplification**
9. **DoS: instability**
10. **DoS: resource consumption (CPU)**
11. **DoS: resource consumption (memory)**
12. **DoS: resource consumption (other)**
13. **Execute unauthorized code or commands**
14. **Gain privileges / assume identity**
15. **Bypass protection mechanism**
16. **Hide activities**

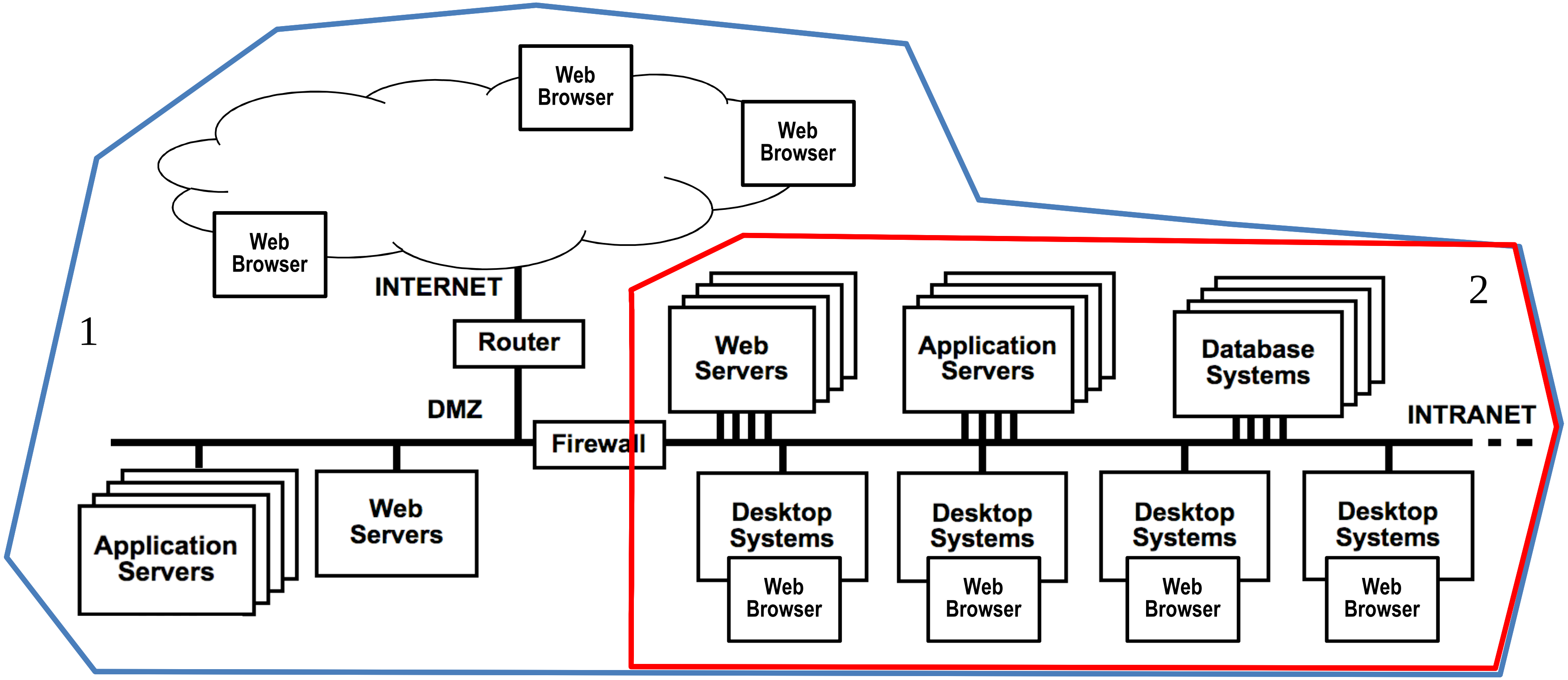
Common Weakness Scoring System (CWSS)

Archetypes:

- Web Browser User Interface
- Web Servers
- Application Servers
- Database Systems
- Desktop Systems
- SSL

Vignettes:

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



CWSS for a Technology Group

50%	• Web Vignette 1 … TI(1), TI(2), TI(3),…	Top N List 1
10%	• Web Vignette 2 … TI(1), TI(2), TI(3),…	Top N List 2
10%	• Web Vignette 3 … TI(1), TI(2), TI(3),…	Top N List 3
10%	• Web Vignette 4 … TI(1), TI(2), TI(3),…	Top N List 4
15%	• Web Vignette 5 … TI(1), TI(2), TI(3),…	Top N List 5
15%	• Web Vignette 6 … TI(1), TI(2), TI(3),…	Top N List 6
	Web Application Technology Group	Top 10 List

CWE Top 10 List for Web Applications can be used to:

- **Identify skill and training needs for your web team**
- **Include in T’s & C’s for contracting for web development**
- **Identify tool capability needs to support web assessment**

Questions?

ramartin@mitre.org