Homeland Security

Commerce

National Defense

BUILDING SECURITY IN
SOFTWARE ASSURANCE

Software Security Knowledge: Training

http://cwe.mitre.org

CWE™

CAPEC™

http://capec.mitre.org

Robert A. Martin
Sean Barnum

May 2011

MITRE

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **MAY 2011**	2. REPORT TYPE	3. DATES COVERED **00-00-2011 to 00-00-2011**
4. TITLE AND SUBTITLE **Software Security Knowledge: Training**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Mitre Corporation,202 Burlington Rd,Bedford,MA,01730-1420**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Presented at the 23rd Systems and Software Technology Conference (SSTC), 16-19 May 2011, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**28**	

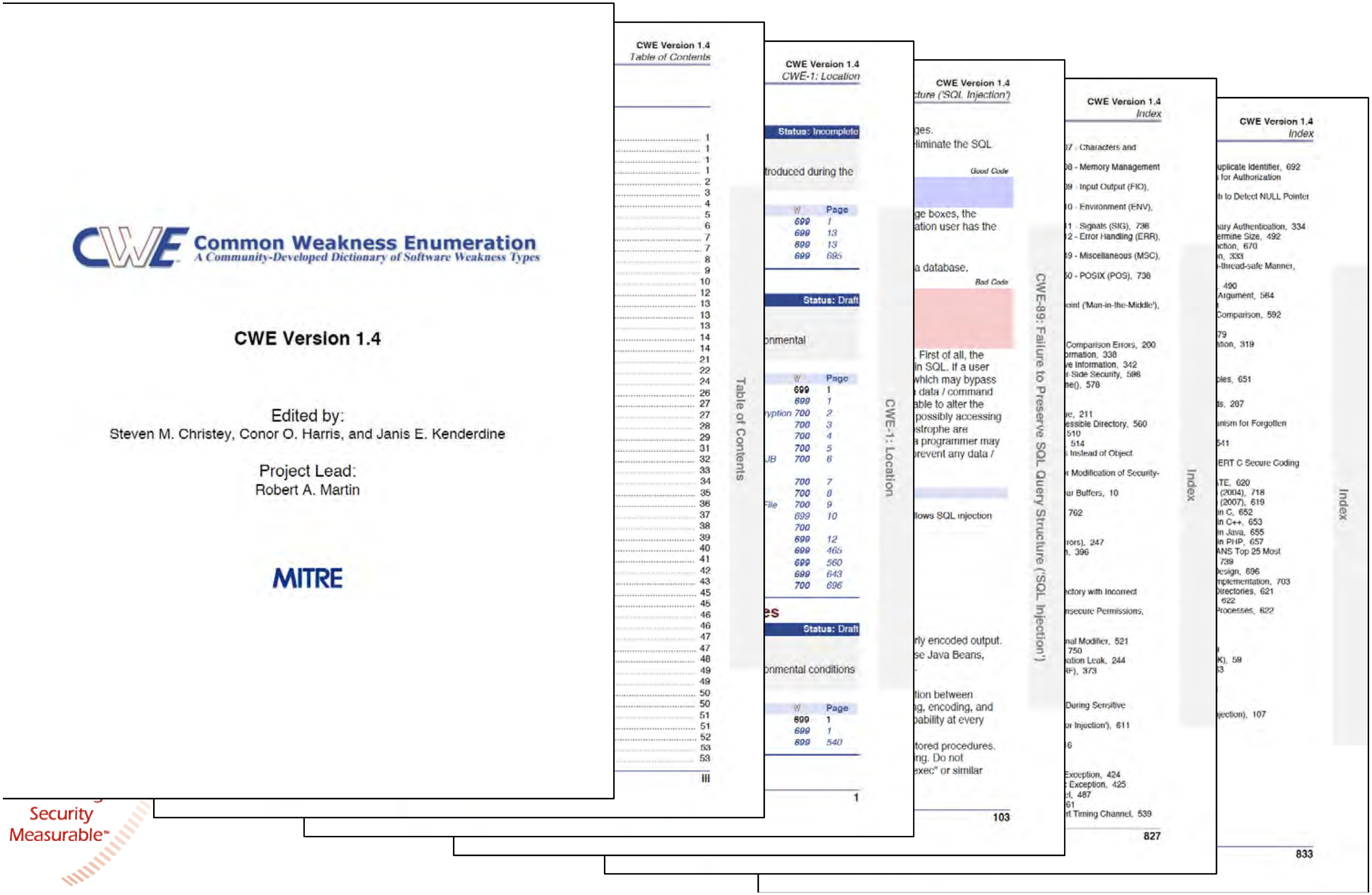
Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Agenda

8:00-8:45am	Software Security Knowledge about Applications Weaknesses
9:00-9:45am	Software Security Knowledge about Attack Patterns Against Applications
	Training in Software Security
10:15-11:00am	Software Security Practice
11:15-12:00am	Supporting Capabilities
	Assurance Cases
	Secure Development & Secure Operations

Making Security Measurable™

CWE is Meant for People to Use



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.)

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.

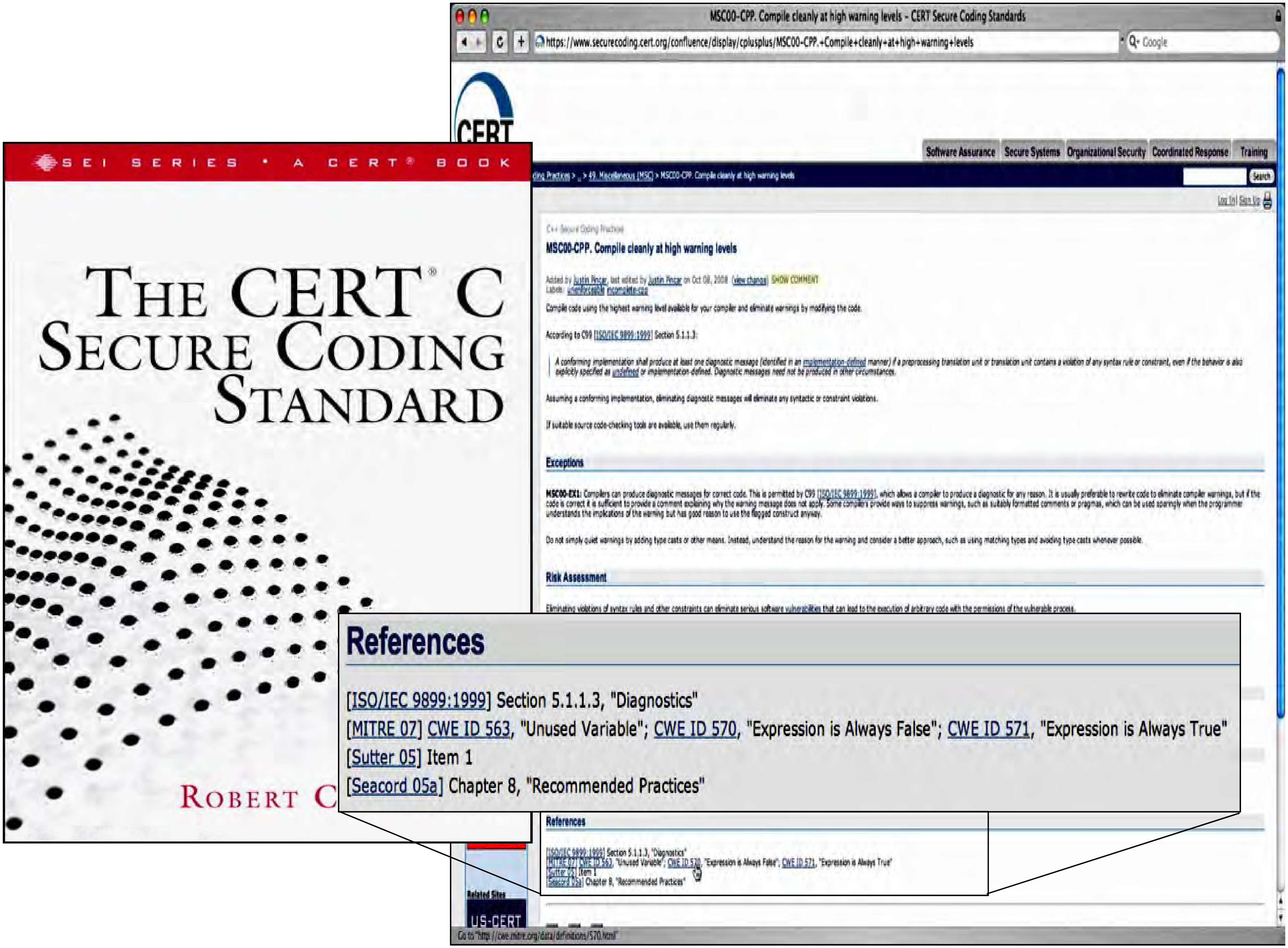
The **Certified Secure Software Lifecycle Professional** (CSSLP) Certification Program will show software lifecycle stakeholders not only how to implement security, but how to glean security requirements, design, architect, test and deploy secure software.

An Overview of the Steps:

(ISC)² ® 5-day CSSLP CBK® Education Program

Educate yourself and learn security best practices and industry standards for the software lifecycle through the CSSLP Education Program.(ISC)² provides education your way to fit your life and schedule.Completing this course will, not only teach all of the

stablish a security plan across your



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
SEI SERIES • A CERT® BOOK
THE CERT® C SECURE CODING STANDARD
ROBERT C
MSC00-CPP. Compile cleanly at high warning levels
Exceptions
Risk Assessment
References
[ISO/IEC 9899:1999] Section 5.1.1.3, "Diagnostics"
[MITRE 07] CWE ID 563, "Unused Variable"; CWE ID 570, "Expression is Always False"; CWE ID 571, "Expression is Always True"
[Sutter 05] Item 1
[Seacord 05a] Chapter 8, "Recommended Practices"

SANS

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

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

SANS MITRE

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 Signatures
  - Use of Less Trusted Source
  - Acceptance of Extraneous Untrusted Data With Trusted Data
  - Improperly Trusted Reverse DNS
  - Insufficient Type Distinction
  - Cross-Site Request Forgery (CSRF) - (352)
  - Failure to Add Integrity Check Value
  - Improper Validation of Integrity Check Value
  - Trust of System Event Data
  - Reliance on File Name or Extension of Externally-Supplied File
  - Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
- Privacy Violation
- Reliance on Cookies without Validation and Integrity Checking
- Client-Side Enforcement of Server-Side Security - (602)
- Improperly Implemented Security Check for Standard
- Improper Authentication
- User Interface Security Issues
- Use of Insufficiently Random Values - (330)
- Logging of Excessive Data
- Certificate Issues
- Cryptographic Issues
  - Key Management Errors
  - Missing Required Cryptographic Step
  - Not Using a Random IV with CBC Mode
  - Failure to Encrypt Sensitive Data
    - Cleartext Storage of Sensitive Information
    - Cleartext Transmission of Sensitive Information - (319)
    - Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
  - Reversible One-Way Hash
  - Inadequate Encryption Strength
    - Use of a Broken or Risky Cryptographic Algorithm - (327)
  - Use of RSA Algorithm without OAEP
- Permissions, Privileges, and Access Controls
  - Access Control (Authorization) Issues - (284)
  - Permission Issues
    - Incorrect Default Permissions
    - Insecure Inherited Permissions
    - Insecure Preserved Inherited Permissions
    - Incorrect Execution-Assigned Permissions
    - Improper Handling of Insufficient Permissions or Privileges
    - Improper Preservation of Permissions
    - Exposed Unsafe ActiveX Method
    - Incorrect Permission Assignment for Critical Resource - (732)
    - Permission Race Condition During Resource Copy
  - Privilege / Sandbox Issues
  - Improper Ownership Management
  - Incorrect User Management
- Password in Configuration File
- Insufficient Compartmentalization
- Reliance on a Single Factor in a Security Decision
- Insufficient Psychological Acceptability
- Reliance on Security through Obscurity
- Protection Mechanism Failure
- Insufficient Logging
- Reliance on Cookies without Validation and Integrity Checking in a Security Decision

Insufficient Encapsulation

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

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.
- .NET Framework Security — Code Review; ~~http://msdn.microsoft.com/en-us/library/aa302437.aspx~~
- *Common Weakness Enumeration*, MITRE; http://cwe.mitre.org/
- ~~Security Code Reviews;~~ http://www.codesecurely.org/Wiki/view.aspx/Security_Code_Reviews
- Security Code Review — Use Visual Studio Bookmarks To Capture Security Findings; http://blogs.msdn.com/aliki/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
- OWASP Top Ten; http://www.owasp.org/index.php/OWASP_Top_Ten_Project

10

SAFE[Code] Driving Secu[rity and Integrity]

Testing

Testing activities validate the secure implementation of a product, which red[uces] the likelihood of security bugs being released and discovered by customers a[nd/or] malicious users. The majority of SAFECode members have adopted the foll[owing] software security testing practices in their software development lifecycle. The [goal] is not to "test in security," but rather to validate the robustness and secur[ity of] the software products prior to making the product available to customers. T[hese] 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 bu[ilding] intentionally malformed data and then having the software under test consume the malformed data to see how it responds. The science of fuzz testing is somewhat new but it is maturing rapidly. There is a small market for fuzz testing tools today, but in many cases software developers must build bespoke fuzz testers to suit specialized file and network data formats. Fuzz testing is an effective testing technique because it uncovers weaknesses in data handling code.

Resources

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

16

SAFECode Driving Security and Integrity

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

Fundamental Practices for
Secure Software Development

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

OCTOBER 8, 2008

LEAD WRITER Michael Howard, Microsoft Corp.

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

CWE

CAPEC

Making Security Measurable™

The Security Development Lifecycle : MS08-078 and the SDL

http://blogs.msdn.com/sdl/archive/2008/12/18/ms08-078-and-the-sdl.aspx

Welcome to MSDN Blogs Sign in | Join | Help

The Security Development Lifecycle

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)

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

MS08-078 and the SDL

Hi, Michael here.

Every bug is an opportunity to learn, and the security update that fixed the data binding bug that affected Internet Explorer users is no exception.

The Common Vulnerabilities and Exposures (CVE) entry for this bug is CVE-2008-4844.

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

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

Background

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

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

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

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

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

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

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

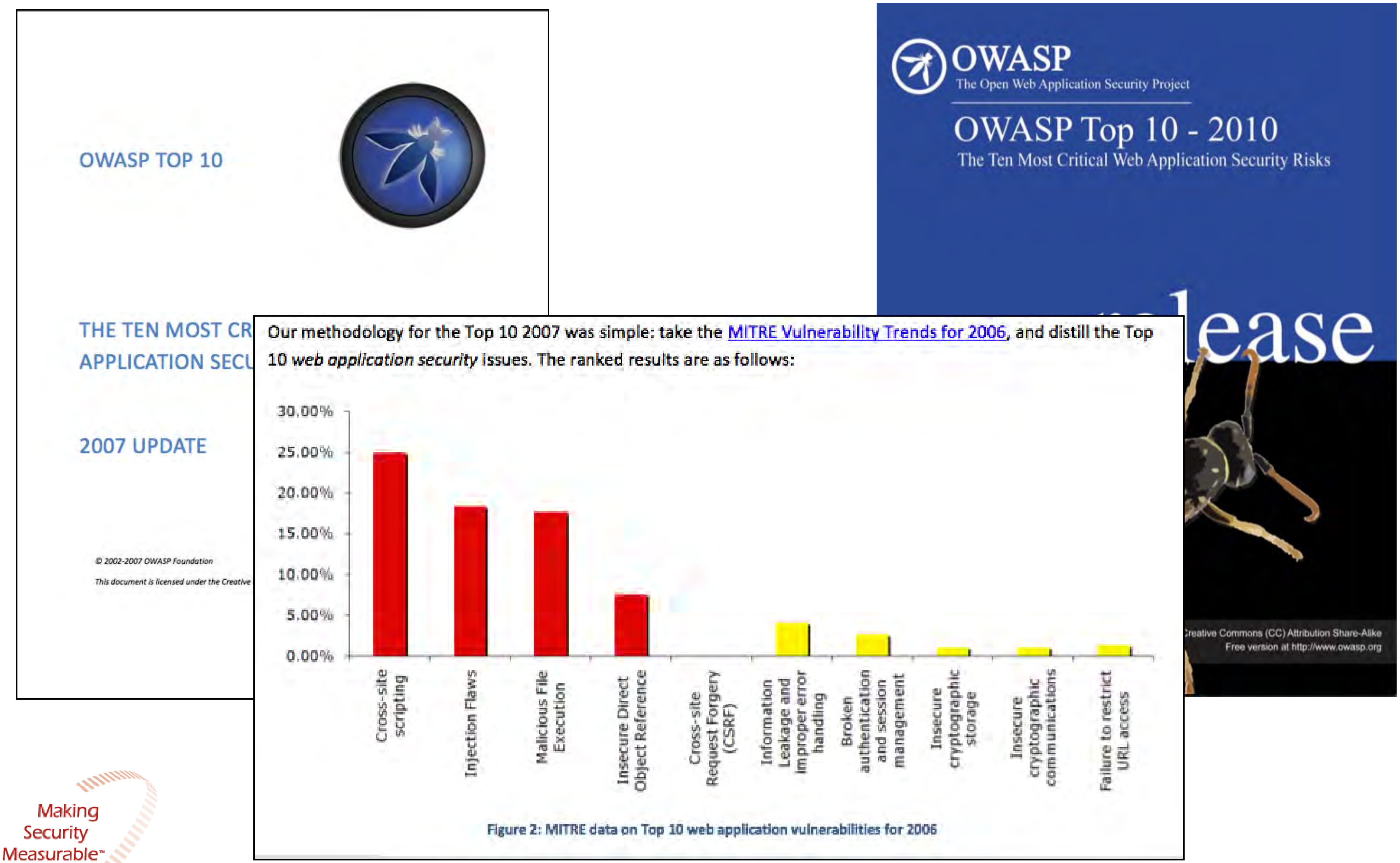
a time-of-check-time-of-use (TOCTOU) bug that led to code calling into a freed memory block. The on Weakness Enumeration (CWE) classification for this vulnerability is CWE-367.

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



Making Security Measurable™

Code Review Introduction - OWASP

http://www.owasp.org/index.php/Code_Review_Introduction

OWASP

Log in

OWASP

The Open Web Application Security Project

Go Search

Page Discussion View source History

Navigation

- Home
- News
- OWASP Projects
- Downloads
- Local Chapters
- Global Committees
- AppSec Job Board
- AppSec Conferences
- Presentations
- Video
- Press
- Get OWASP Books
- Get OWASP Gear
- Mailing Lists
- About OWASP
- Membership

Reference

- How To...
- Principles
- Threat Agents
- Attacks
- Vulnerabilities
- Controls
- Activities
- Technologies
- Glossary
- Code Snippets
- .NET Project
- Java Project

Language

- English
- Español

Code Review Introduction

««Code Review Guide History«« | Main (Table of Contents) | »»Preparation»»

Contents [hide]

Introduction

Code review is probably the single-most effective technique for identifying security flaws. When used together with automated tools and manual penetration testing, code review can significantly increase the cost effectiveness of an application security verification effort.

This guide does not prescribe a process for performing a security code review. Rather, this guide focuses on the mechanics of reviewing code for certain vulnerabilities, and provides limited guidance on how the effort should be structured and executed. OWASP intends to develop a more detailed process in a future version of this guide.

Manual security code review provides insight into the "real risk" associated with insecure code. This is the single most important value from a manual approach. A human reviewer can understand the context for certain coding practices, and make a serious risk estimate that accounts for both the likelihood of attack and the business impact of a breach.

Why Does Code Have Vulnerabilities?

MITRE has catalogued almost 700 different kinds of software weaknesses in their CWE project. These are all different ways that software developers can make mistakes that lead to insecurity. Every one of these weaknesses is subtle and many are seriously tricky. Software developers are not taught about these weaknesses in school and most do not receive any training on the job about these problems.

These problems have become so important in recent years because we continue to increase connectivity and to add technologies and protocols at a shocking rate. Our ability to invent technology has seriously outstripped our ability to secure it. Many of the technologies in use today simply have not received any security scrutiny.

There are many reasons why businesses are not spending the appropriate amount of time on security. Ultimately, these reasons stem from an underlying problem in the software market. Because software is essentially a black-box, it is extremely difficult to tell the difference between good code and insecure code. Without this visibility, buyers won't pay more for secure code, and vendors would be foolish to spend extra effort to produce secure code.

One goal for this project is to help software buyers gain visibility into the security of software and start to effect change in the software market.

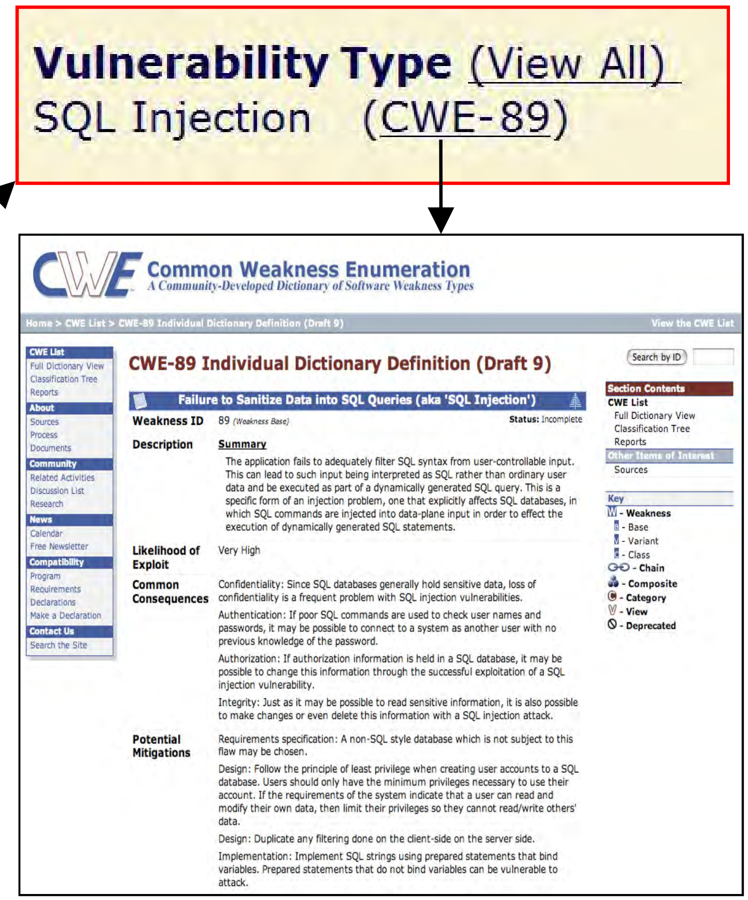
Nevertheless, we still frequently get pushback when we advocate for security code review. Here are some of the (unjustified) excuses that we hear for not putting more effort into security:

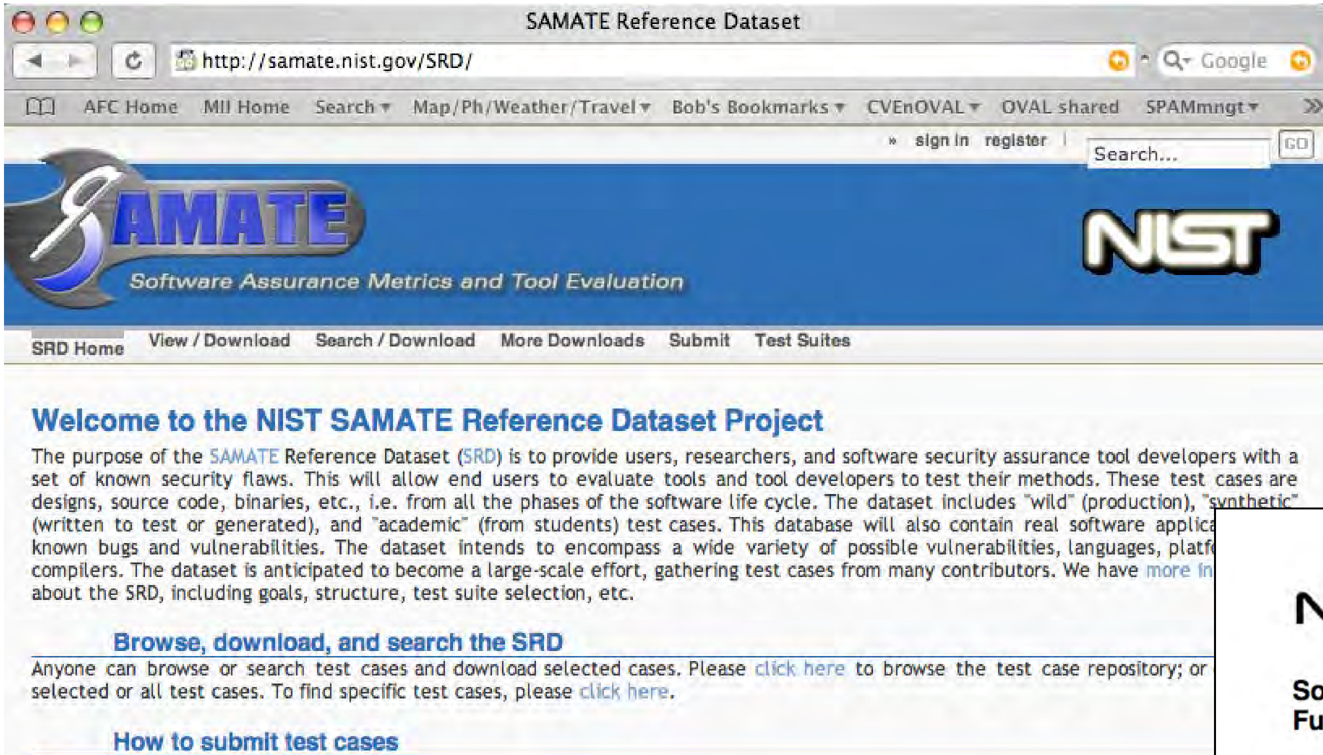
"We never get hacked (that I know of), we don't need security"

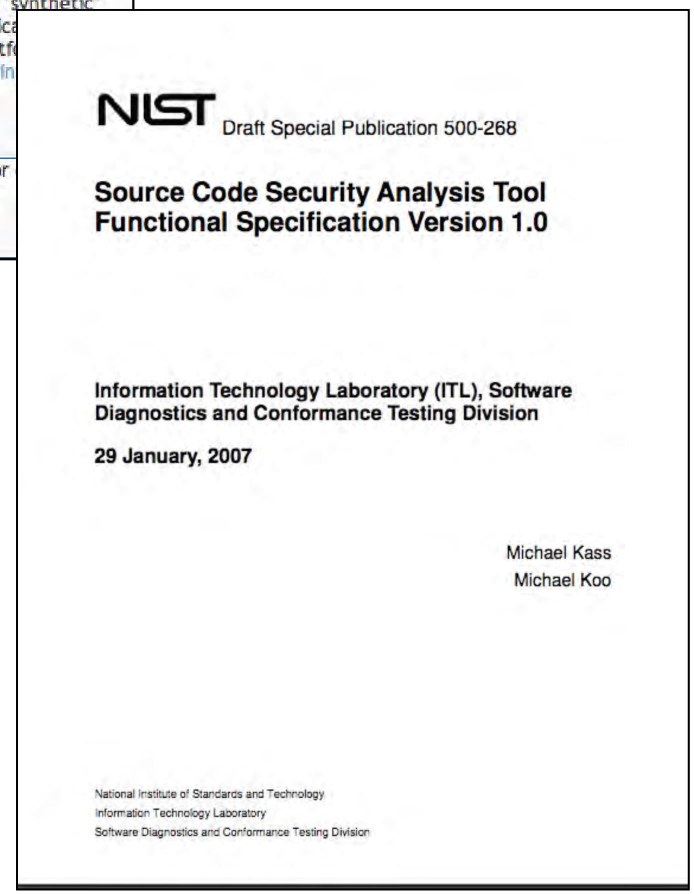
Done

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

NVD XML feeds also include CWE






NIST Draft Special Publication 500-268

Source Code Security Analysis Tool Functional Specification Version 1.0

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

29 January, 2007

Michael Kass
Michael Koo

National Institute of Standards and Technology
Information Technology Laboratory
Software Diagnostics and Conformance Testing Division

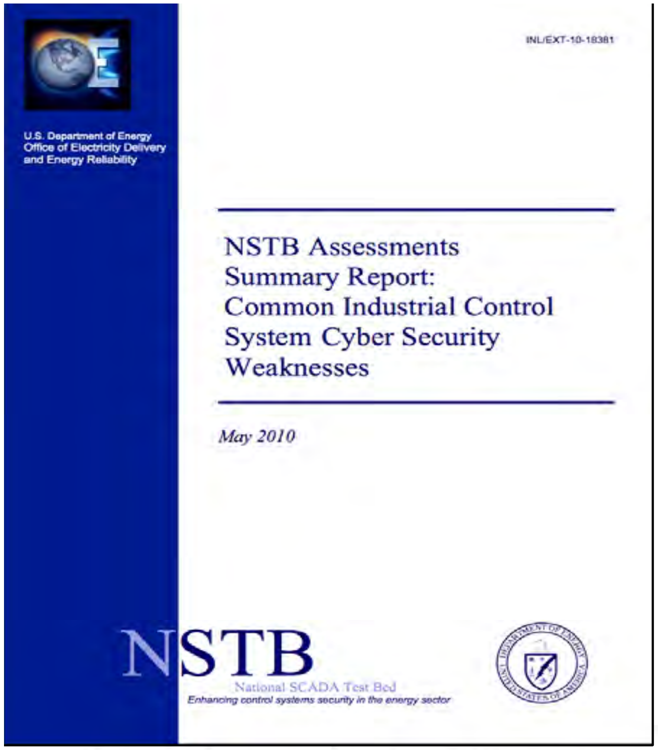
NIST Special Publications:

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

NIST Interagency Reports:

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

Idaho National Labs SCADA Report


INL/EXT-10-18381

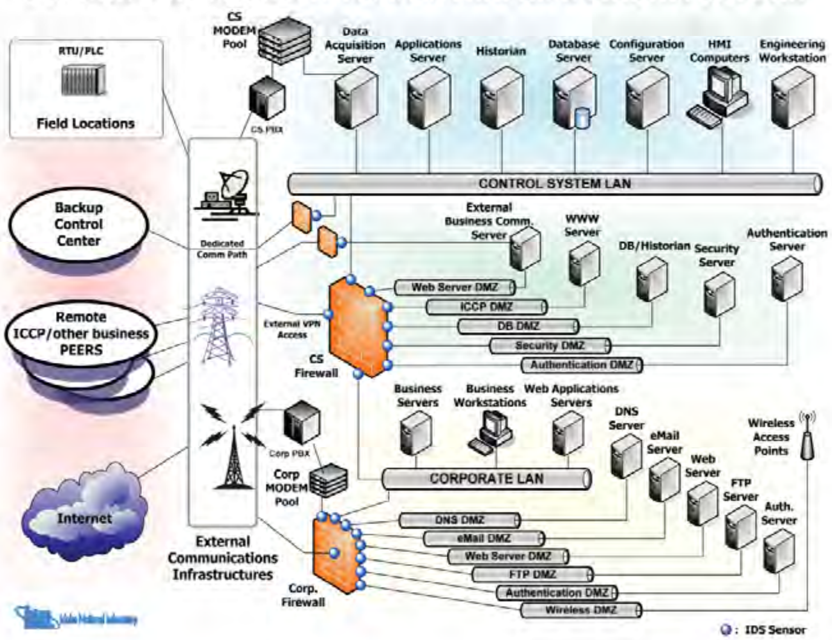
U.S. Department of Energy
Office of Electricity Delivery
and Energy Reliability

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

SECURE CONTROL SYSTEM/ENTERPRISE ARCHITECTURE



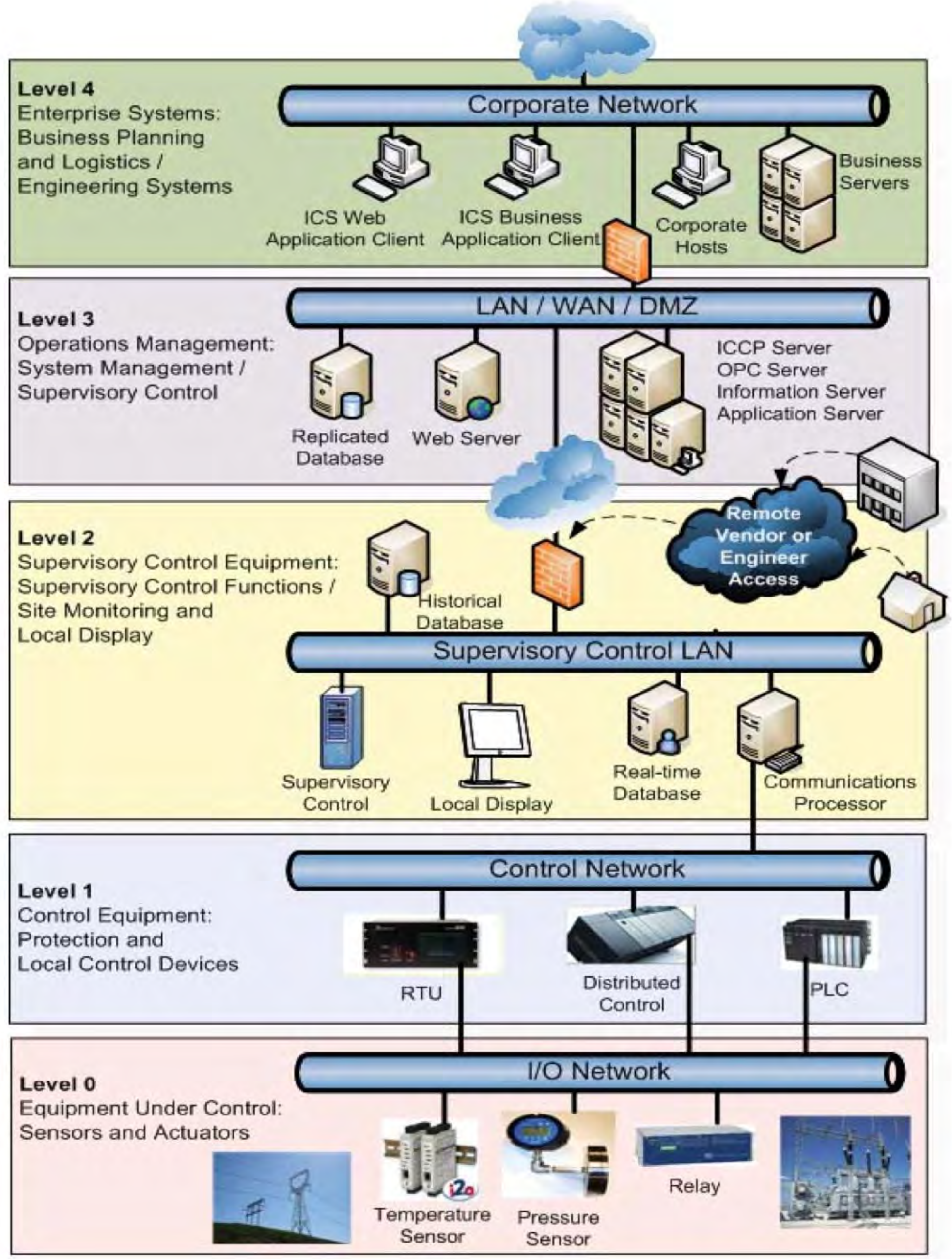


Table 27. Most common programming errors found in ICS code.

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

Top25 | AppSec Street Fighter – SANS Institute

http://blogs.sans.org/appsecstreetfighter/category/top25/

Top 25 Series – Summary and Links

2

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

Done

Welcome to MSDN Blogs Sign in | Join | Help

Search

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 V
116	Improper Encodir 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

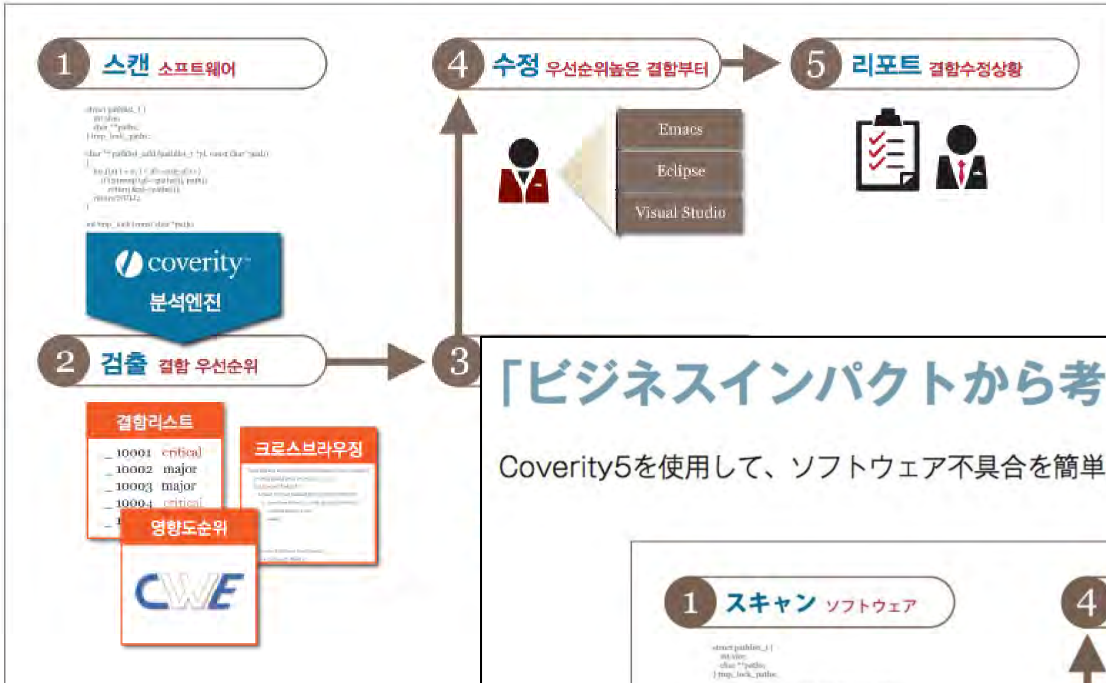
CWE Outreach: A Team Sport

May/June Issue of IEEE Security & Privacy…


armorize
The Web Malware Experts
We speak CWE cwe.mitre.org
We speak CWE cwe.mitre.org
We speak CVE cve.mitre.org
GLOBAL INFORMATION ASSURANCE CERTIFICATION
GIAC
SANS
We speak CWE cwe.mitre.org
coverity
Coverity + Armorize
int val; };
my_struct *ptr, int x, int y)
) && (x < y) || (ptr->val > 0))
ptr->val;
coverity

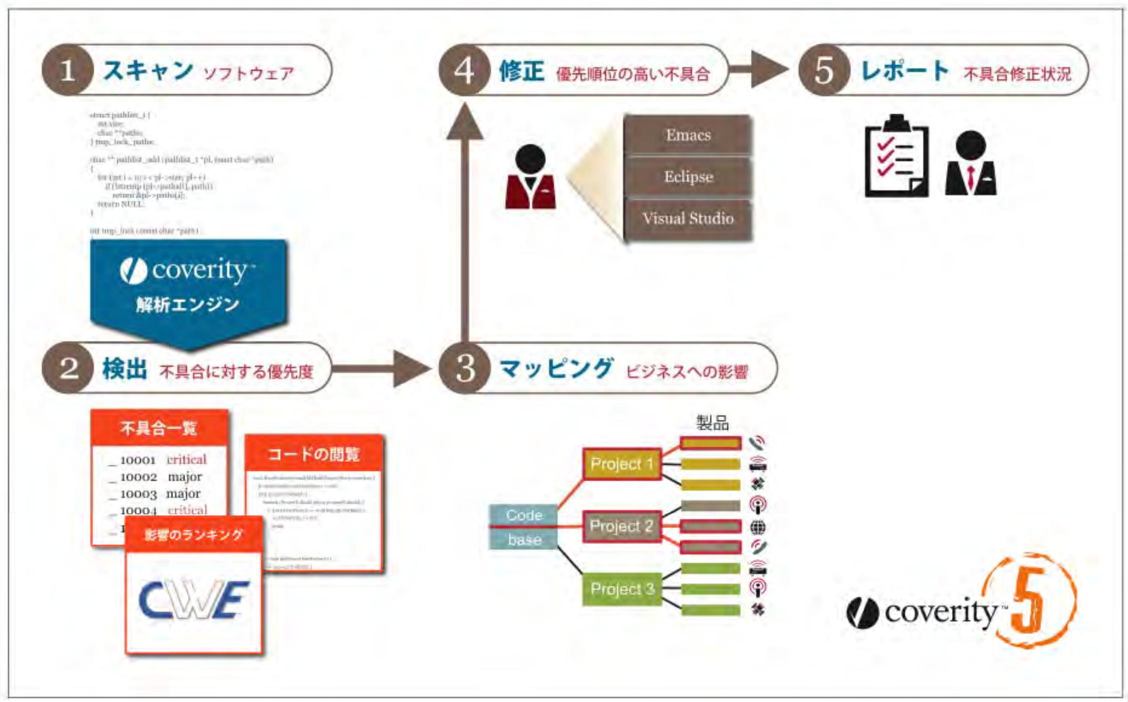
[비즈니스 임팩트를 줄여주는 새로운 품질 관리 방법론]

y5를 사용하여, 소프트웨어 결함을 없애는 5가지 스텝은 아래와 같습니다.



「ビジネスインパクトから考える新しい品質管理」

Coverity5を使用して、ソフトウェア不具合を簡単に除去する 5ステップは以下の通りです。



Korean

Making Security Measurable™

Japanese

Threat Classification Taxonomy Cross Reference View

last edited by Robert Auger 10 months, 3 weeks ago

Page history

Threat Classification 'Taxonomy Cross Reference View'

This view contains a mapping of the WASC Threat Classification's Attacks and Weaknesses with MITRE's Common Weakness Enumeration, MITRE's Common Attack Pattern Enumeration and Classification, OWASP Top Ten 2010 RC1 (original mapping with OWASP Top Ten from Jeremiah Grossman & Bill Corry) and SANS/CWE and OWASP Top Ten 2007 and 2004 (original mapping from Dan Cornell, Denim Group)

WASC ID	Name	CWE ID	CAPEC ID	SANS/CWE Top 25 2009	OWASP Top Ten 2010	OWASP Top Ten 2007	OWASP Top Ten 2004
WASC-01	Insufficient Authentication	287		642	A3 – Broken Authentication and Session Management, A4 – Insecure Direct Object References	A7 – Broken Authentication and Session Management, A4 – Insecure Direct Object Reference	A3 – Broken Authentication and Session management, A2 – Broken Access Control
WASC-02	Insufficient Authorization	284		285	A4 – Insecure Direct Object References, A7 – Failure to Restrict URL Access	A10 – Failure to Restrict URL Access, A4 – Insecure Direct Object Reference	A2 – Broken Access Control
WASC-03	Integer Overflows	190	128	682			
WASC-04	Insufficient Transport Layer Protection	311 523		319	A10 – Insufficient Transport Layer Protection	A9 – Insecure Communications	
WASC-05	Remote File Inclusion	98	193 253	426		A3 – Malicious File Execution	
WASC-06	Format String	134	67				
WASC-07	Buffer Overflow	119 120	10 100	119			A5 – Buffer Overflows
WASC-08	Cross-site Scripting	79	18 19 63	79	A2 – Cross-Site Scripting	A1 – Cross Site Scripting (XSS)	A4 – Cross Site Scripting (XSS)
WASC-09	Cross-site Request Forgery	352	62	352	A5 – Cross-Site Request Forgery	A5 – Cross Site Request Forgery (CSRF)	
WASC-10	Denial of Service	400	119	404	A7 – Failure to Restrict	A10 – Failure to	A9 – Denial of

Tags: Threat Classification
Check for plagiarism

SideBar

WASC Projects

- Distributed Open Proxy Honeypots
- Script Mapping
- The Web Security Glossary
- Web Application Firewall Evaluation Criteria
- Web Application Security Scanner Evaluation Criteria
- Web Application Security Statistics
- Web Hacking Incidents Database
- WASC Threat Classification

WASC Project Leaders

- Robert Auger
- Ryan Barnett
- Romain Gaucher
- Sergey Gordeychik
- Ofer Shezaf
- Brian Shura

WASC Main Website

- http://www.webappsec.org/

WASC Mailing Lists

- http://lists.webappsec.org/

WASC on Twitter

- http://twitter.com/wascupdates

Join us on Linkedin!

- http://www.linkedin.com/groups?gid=83336

Recent Activity

Insufficient Data Protection Working
edited by Robert Auger

Done

IBM Software
Technical White Paper

One way to improve software security is to gain a better understanding of the most common weaknesses that can affect software security. With that in mind, there are many resources available online to help organizations learn about

Resources available to help organizations protect systems in

Resource	Focus
DoD Information Assurance Certification and Accreditation Process (DIACAP)	The DIACAP defines the minimum standa accredited by the DoD and authorized to application-level security controls, but it i activities, general tasks, and a managem
Defense Information Systems Agency (DISA)	The DISA provides a security technical in development that offer more granular info bility assessment techniques. The checklist is the same one used by DoD auditors.
U.S. Department of Homeland Security (DHS)	The DHS offers information on security best practices and tools for application- and soft part of its "Build Security In" initiative.
The Common Weaknesses Enumeration project, a community-based program sponsored by the MITRE Corporation, an IBM Business Partner	The MITRE Corporation maintains the online common vulnerabilities and exposures (CVE enumeration (CWE) knowledge bases about currently known vulnerabilities and types of knowledge base focuses on packaged software and deals with patches and known vul knowledge base focuses on code vulnerabilities.
The Open Web Application Security Project (OWASP)	One of the best sources for information on web application security issues, the OWASP 10 list of the most dangerous and most commonly found and commonly exploited vulne how to identify, fix and avoid them.
Cigital Building Security In Maturity Model (BSIMM)	Created by Cigital, an IBM Business Partner, the BSIMM is designed to help organizatio and plan a software security initiative. The focus is on making applications more secure, process and at later stages in the software life cycle.
IBM X-Force™ research and development team	A global cyberthreat and risk analysis team that monitors traffic and attacks around the IBM X-Force team is an excellent resource for trend analysis and answers to questions attacks are most common, where they are coming from and what organizations can do the risks.
IBM Institute for Advanced Security (IAS)	This companywide cybersecurity initiative applies IBM research, services, software and t help governments and other clients improve the security and resiliency of their IT and bu

Test and vulnerability assessment

Testing applications for security defects should be an integral and organic part of any software testing process. During security testing, organizations should test to help ensure that the security requirements have been implemented and the product is free of vulnerabilities.

The SEF refers to the MITRE Common Weakness Enumeration[5] (CWE) list and the Common Vulnerability E be tested. Thi information ar and vulnerabi against the m

Creating a se plan includes

[5] For more inforr
[6] For more inforr

Security in Development: The IBM Secure Engineering Framework

Redguides for Business Leaders

Danny Allan
Tim Hahn
Andras Szakal
Jim Whitmore
Axel Buecker

- Investigating common development processes and the IBM Integrated Product Development process
- Emphasizing security awareness and requirements in the software development process
- Discussing test and vulnerability assessments

Redbooks

Software Engineering Institute

Making the Business Case for Software Assurance

Nancy R. Mead
Julia H. Allen
W. Arthur Conklin
Antonio Drommi
John Harrison
Jeff Ingalsbe
James Rainey
Dan Shoemaker

April 2009

SPECIAL REPORT
CMU/SEI-2009-SR-001

CERT Program

http://www.sei.cmu.edu

Carnegie Mellon

Security Measurable™

OVM: An Ontology for Vulnerability Management

Ju An Wang & Minzhe Guo
Southern Polytechnic State University
1100 South Marietta Parkway
Marietta, GA 30060
(01) 678-915-3718
jwang@spsu.edu

ABSTRACT

In order to reach the goals of the Information Security Automation Program (ISAP) [1], we propose an ontological approach to capturing and utilizing the fundamental concepts in information security and their relationship, retrieving vulnerability data and reasoning about the cause and impact of vulnerabilities. Our ontology for vulnerability management (OVM) has been populated with all vulnerabilities in NVD [2] with additional inference rules, knowledge representation, and data-mining mechanisms. With the seamless integration of common vulnerabilities and their related concepts such as attacks and countermeasures, OVM provides a promising pathway to making ISAP successful.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General [Security and protection]; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection;

General Terms

Ontology, Security, Vulnerability Analysis and Management

Keywords

Security vulnerability, Semantic technology, Ontology, Vulnerability analysis

1. INTRODUCTION

The Information Security Automation Program (ISAP) is a U.S. government multi-agency initiative to enable automation and standardization of technical security operations [1]. Its high-level goals include standards based automation of security checking and remediation as well as automation of technical compliance activities. Its low-level objectives include enabling standards based communication of vulnerability data, customizing and managing configuration baselines for various IT products, assessing information systems and reporting compliance status, using standard metrics to weight and aggregate potential vulnerability impact, and remediating identified vulnerabilities [1]. Secure computer systems ensure that confidentiality, integrity, and availability are maintained for users, data, and other information assets. Over the past a few decades, a significantly large amount of knowledge has been accumulated in the area of information security. However, a lot of concepts in information security are vaguely defined and sometimes they have different semantics in different contexts, causing misunderstanding among stake holders due to the language ambiguity. On the other hand, the standardization, design and development of security tools [1-5] require a systematic classification and definition of security concepts and techniques. It is important to have a clearly defined vocabulary and standardized language as means to accurately communicate system vulnerability information and their countermeasures among all the people involved. We believe that semantic technology in general, and ontology in particular, could be a useful tool for system security. Our research work has confirmed this belief and this paper will report some of our work in this area.

An ontology is a specification of concepts and their relationship. Ontology represents knowledge in a formal and structured form. Therefore, ontology provides a better tool for communication, reusability, and organization of knowledge. Ontology is a knowledge representation (KR) system based on Description Logics (DLs) [6], which is an umbrella name for a family of KR formalisms representing knowledge in various domains. The DL formalism specifies a knowledge domain as the "world" by first defining the relevant concepts of the domain, and then it uses these concepts to specify properties of objects and individuals occurring in the domain [10-12]. Semantic technologies not only provide a tool for communication, but also a foundation for high-level reasoning and decision-making. Ontology, in particular, provides the potential of formal logic inference based on well-defined data and knowledge bases. Ontology captures the relationships between collected data and use the explicit knowledge of concepts and relationships to deduce the implicit and inherent knowledge. As a matter of fact, a heavy-weight ontology could be defined as a formal logic system, as it includes facts and rules, concepts, concept taxonomies, relationships, properties, axioms and constraints.

A vulnerability is a security flaw, which arises from computer system design, implementation, maintenance, and operation. Research in the area of vulnerability analysis focuses on discovery of previously unknown vulnerabilities and quantification of the security of systems according to some metrics. Researchers at MITRE have provided a standard format for naming a security vulnerability, called Common Vulnerabilities and Exposures (CVE) [14], which assigns each vulnerability a unique identification number. We have designed a vulnerability ontology OVM (ontology for vulnerability management) populated with all existing vulnerabilities in NVD [2]. It supports research on reasoning about vulnerabilities and characterization of vulnerabilities and their impact on computing systems. Vendors and users can use our ontology in support of vulnerability analysis, tool development and vulnerability management.

The rest of this paper is organized as follows: Section 2 presents the architecture of our OVM. Section 3 discusses how to populate the OVM with vulnerability instances from NVD and other

CSIIRW '09, April 13-15, Oak Ridge, Tennessee, USA

A Policy-Based Vulnerability Analysis Framework

By

SOPHIE JEAN ENGLE
B.S. (University of Nebraska at Omaha) 2002

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Professor Matt Bishop (Chair)

Professor S. Felix Wu

Professor Karl Levitt

Professor Sean Peisert

Committee in Charge

2010

i

Analysis-Based Verification: A Programmer-Oriented Approach to the Assurance of Mechanical Program Properties

T. J. Halloran
May 27, 2010
CMU-ISR-10-112

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
William L. Scherlis (advisor)
James D. Herbsleb
Mary Shaw
Joshua J. Bloch, Google, Inc.

This material is based upon work supported by the following grants: NASA: NCC2-1298 and NNA05C530A; Lockheed Martin: RRMHS1798; ARO: DAAD190210389; IBM Eclipse: IC-5010. The views expressed in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied of the sponsor, the U.S. Government, or Carnegie Mellon University.

Linkage with Fundamental Changes in Enterprise Security Initiatives

Twenty Critical Controls for Effective Cyber Def 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
- Critical Control 2: Inventory of Authorized and Unauthorized
- Critical Control 3: Secure Configurations for Hardware and So Servers
- Critical Control 4: Secure Configurations for Network Devices
- Critical Control 5: Boundary Defense
- Critical Control 6: Maintenance, Monitoring, and Analysis of
- Critical Control 7: Application Software Security
- Critical Control 8: Controlled Use of Administrative Privilege
- Critical Control 9: Controlled Access Based on Need to Know
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr
- Critical Contr

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

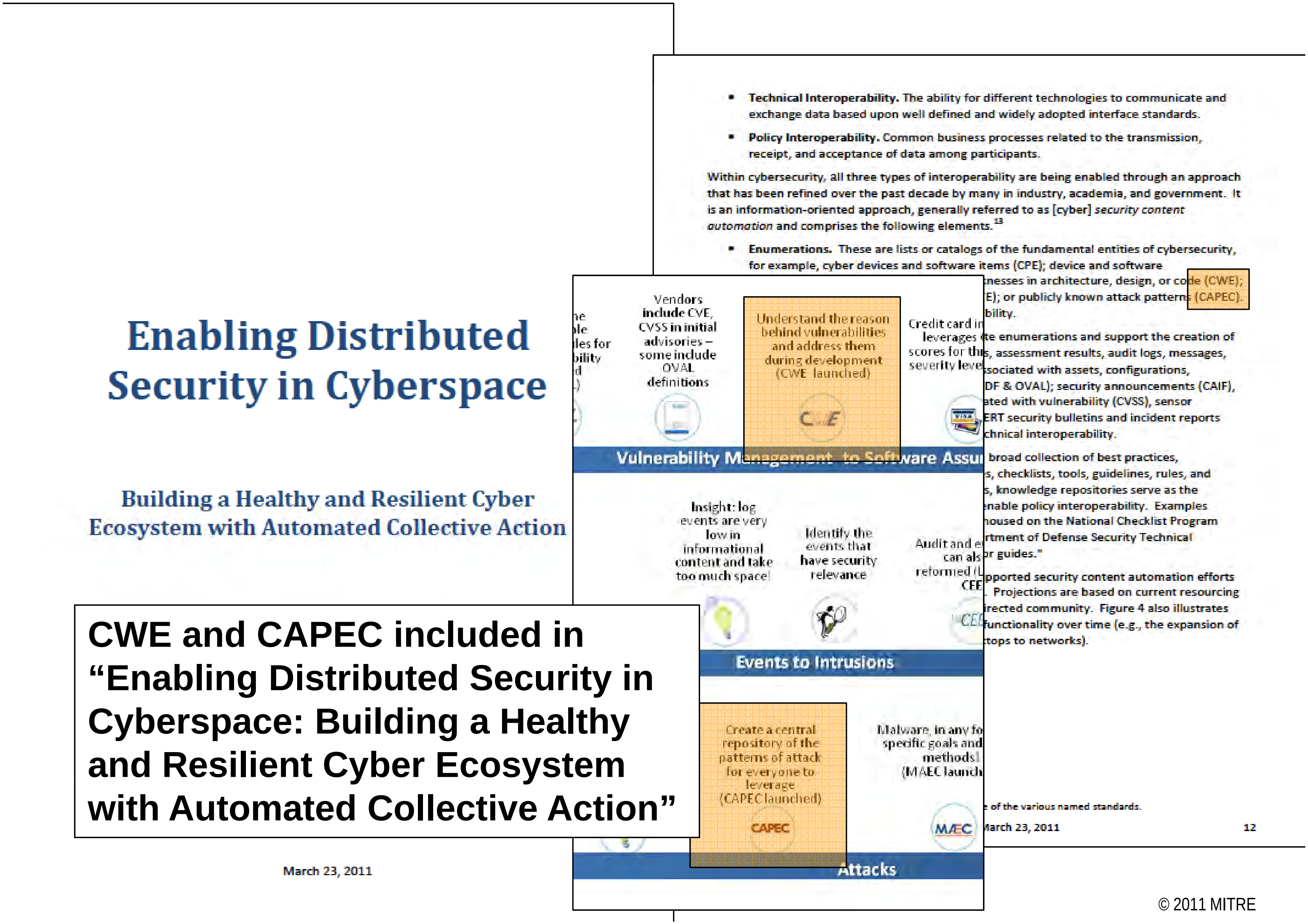
CWE and CAPEC included in Control 7 of the "Twenty Critical Controls for Effective Cyber Defense: Consensus Audit Guidelines"

Procedures and tools for implementing t

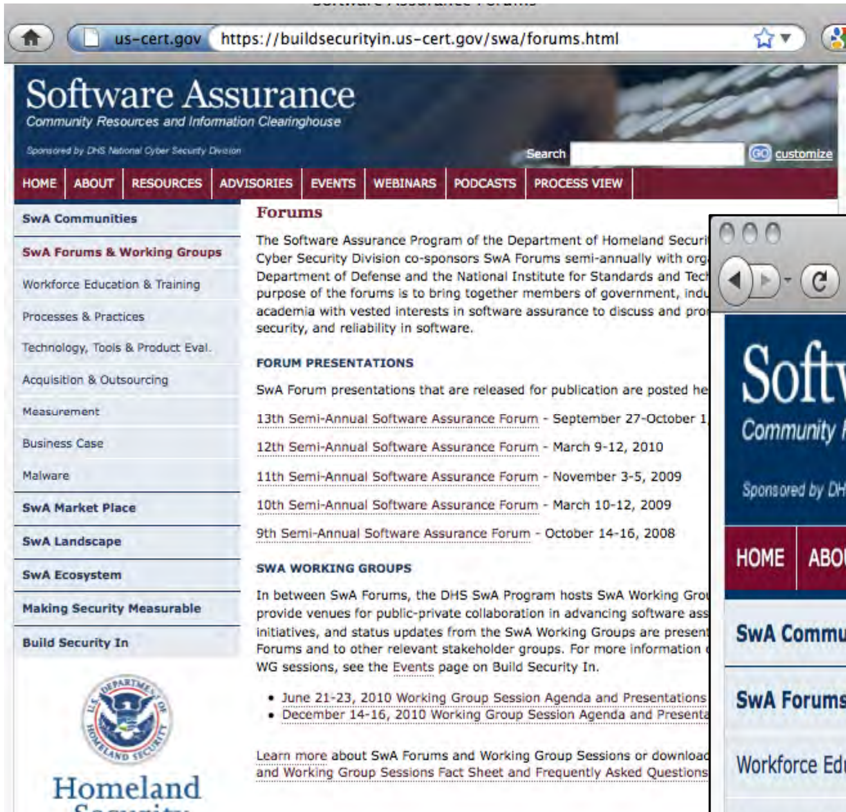
Source code testing tools, web application security scanning tools, and object code testing tools have proven useful in securing application software, along with manual application security penetration testing by testers who have extensive programming knowledge as well as application penetration testing 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.

Making Security Measurable™

Linkage with Fundamental Changes in Enterprise Security Initiatives



CWE and CAPEC included in "Enabling Distributed Security in Cyberspace: Building a Healthy and Resilient Cyber Ecosystem with Automated Collective Action"


us-cert.gov https://buildsecurityin.us-cert.gov/swa/forums.html
Software Assurance
Community Resources and Information Clearinghouse
Search
customize
HOME ABOUT RESOURCES ADVISORIES EVENTS WEBINARS PODCASTS PROCESS VIEW
SwA Communities
SwA Forums & Working Groups
Workforce Education & Training
Processes & Practices
Technology, Tools & Product Eval.
Acquisition & Outsourcing
Measurement
Business Case
Malware
SwA Market Place
SwA Landscape
SwA Ecosystem
Making Security Measurable
Build Security In
Homeland
Forums
FORUM PRESENTATIONS
12th Semi-Annual Software Assurance Forum - March 9-12, 2010
11th Semi-Annual Software Assurance Forum - November 3-5, 2009
10th Semi-Annual Software Assurance Forum - March 10-12, 2009
9th Semi-Annual Software Assurance Forum - October 14-16, 2008
SWA WORKING GROUPS
June 21-23, 2010 Working Group Session Agenda and Presentations


We speak
CVE
cve.mitre.org
We speak
CWE
cwe.mitre.org
We use
OVAL
We know
MAEC
maec.mitre.org
U.S. CERT
Homeland Security


Software Assurance Technology, Tools and Product Evaluation Working Group Resources
us-cert.gov https://buildsecurityin.us-cert.gov/swa/ttperesrc.html
Software Assurance
Community Resources and Information Clearinghouse
Sponsored by DHS National Cyber Security Division
Search
customize
HOME ABOUT RESOURCES ADVISORIES EVENTS WEBINARS PODCASTS PROCESS VIEW
SwA Communities
SwA Forums & Working Groups
Workforce Education & Training
Processes & Practices
Technology, Tools & Product Eval.
- Activities
- Resources
- Collaborations
- Research
Acquisition & Outsourcing
Measurement
Business Case
Technology, Tools and Product Evaluation Working Group
Resources
Build Security In
SwA Tools Overview
CERT Secure Coding Standards
Common Attack Pattern Enumeration and Classification (CAPEC)
Common Weakness Enumeration (CWE)
The Data & Analysis Center for Software
Federal Plan for Cyber Security and Information Assurance Research and Development: Available for download on the National Coordination Office for Networking and Information Technology Research and Development site.
Function Extraction: Automated Behavior Computation for Aerospace Software Verification and Certification (PDF)
ISO/IEC SC22 OWGV Guidance for Avoiding Vulnerabilities through Language Selection and Use
Done

Questions?

ramartin@mitre.org