



Homeland
Security



Commerce



National
Defense



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	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
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 Same as Report (SAR)	18. NUMBER OF PAGES 28	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

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

Security
Measurable™Index

16 July 2010

A Human Capital Crisis in Cybersecurity

Technical Proficiency Matters

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

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

PROJECT DIRECTOR
J. Edgar Hoover

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.

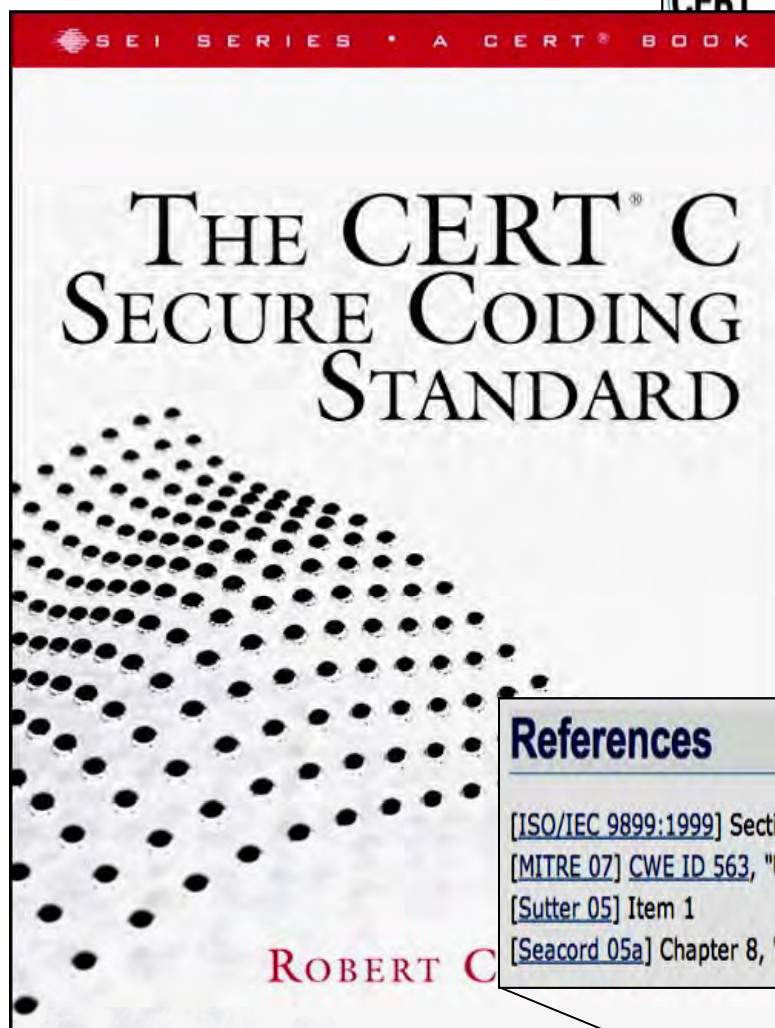
An Overview of the Steps:

(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

establish 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

CERT

Software Assurance Secure Systems Organizational Security Coordinated Response Training

Log In! Sign Up

49. Miscellaneous [MSC] > MSC00-CPP. Compile cleanly at high warning levels

C++ Secure Coding Practices

MSC00-CPP. Compile cleanly at high warning levels

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

Labels: [uncomfortable](#) [incomplete-cpp](#)

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

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

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

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

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

Exceptions

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

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

Risk Assessment

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

References

[ISO/IEC 9899:1999] Section 5.1.1.3, "Diagnostics"

[MITRE 07] [CWE ID 563](#), "Unused Variable"; [CWE ID 570](#), "Expression is Always False"; [CWE ID 571](#), "Expression is Always True"

[Sutter 05] Item 1

[Seacord 05a] Chapter 8, "Recommended Practices"

References

[ISO/IEC 9899:1999] Section 5.1.1.3, "Diagnostics"

[MITRE 07] [CWE ID 563](#), "Unused Variable"; [CWE ID 570](#), "Expression is Always False"; [CWE ID 571](#), "Expression is Always True"

[Sutter 05] Item 1

[Seacord 05a] Chapter 8, "Recommended Practices"

Related Sites

US-CERT

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

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 Enhancement
- Expected Behavior Violation

Initialization and Cleanup Errors

- Insecure Default Variable Initialization
- External Initialization of Trusted Variables
- Non-volt on Failed Initialization
- Missing Initialization
- Incomplete Cleanup
- Improper Cleanup on Thrown Exception
- Improper Initialization (186)

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 size off on a Pointer Type
- Incorrect Pointer Scaling
- Use of Pointer Subtraction to Determine Size
- Assignment of a Fixed Address to a Pointer
- Attempt to Access Child of a Non-structure Pointer

Time and State

- State Issues
- Incomplete Internal State Destruction
- State Synchronization Error
- Mutable Objects Passed by Reference
- Passing Mutable Objects to an Untrusted Method
- Removal of Critical State Data (188)

Basic Condition (182)

- Session Fixation
- Concurrency Issues
- Temporary File Issues
- Content Timing Channel
- Technology Specific Time and State Issues
- Synthetic Name not Mapping to Correct Object
- Signal Errors
- Unrestricted Externally Accessible Lock
- Double-Checked Locking
- Insufficient Session Expiration
- Insufficient Synchronization
- Use of a Non-mutually Exclusive Function in an Unmutually Exclusive 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-obliguous 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 (304)
- Empty Synchronized Block
- Explicit Call to Finalize()
- Reachable Assertion
- Use of Potentially Dangerous Function

Security Features

Credentials Management

- Hard-Coded Password (309)
- Unverified Password Change
- Missing Password Field Masking
- Weak Cryptography for Passwords
- Weak Password Requirements
- Not Using Password Aging
- Password Aging with Long Expiration
- Unaffordably Protected Credentials
- Weak Password Recovery Mechanism for Forgotten Password

Insufficient Verification of Data Authenticity

- Origin Validation Error
- Improper Verification of Cryptographic Signatures
- Use of Low Trusted Source
- Acceptance of Extraneous Untrusted Data With Trusted Data
- Improperly Trusted Remote DNS
- Insufficient Type Distinction
- Clear Site Request Forgery (CSRF) (253)
- Failure to Add Integrity Check Value
- Improper Validation of Integrity Check Value
- Trust of System Event Data
- Reliance on File Name or Extension on 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
- Power-Side Exploitation of Secure-Side Security (146)
- 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
- Untrusted Storage of Sensitive Information
- Untrusted Transmission of Sensitive Information (316)
- Session Cookie in HTTPS Session Without 'Secure' Attribute
- Reversible One-Way Hash
- Inadequate Encryption Strength
- Use of a Broken or Flawed Cryptographic Algorithm (175)
- Use of RSA Algorithm without OAEP

Permissions, Privileges, and Access Controls

- Access Control (Mis)configuration Issues (314)
- Permission Issues
- Incorrect Default Permissions
- Insecure Inherited Permissions
- Insecure Preserved Inherited Permissions
- Incorrect Execution Assigned Permissions
- Improper Handling of Inconsistent Permissions of Privileges
- Improper Preservation of Permissions
- Exposed Unsafe Access Method
- Insecure Permissions 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 (Executable) Method Without Real (Object) Check
- Use of Inner Class Containing Sensitive Data
- Global Public Variable Without Final Modifier
- Downloaded Code Without Integrity Check (404)
- Array Declared Public, Final, and Static
- Analysis Method Declared Public

Leftover Debug Code

- Use of Dynamic Class Loading
- clone() Method Without super.clone()
- Comparison of Classes by Name
- Data Leak Between Sessions
- Trust Boundary Violation

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

Data Handling

Numeric Errors

- Use of Incorrect Byte Ordering
- Unchecked Array Indexing
- Incorrect Conversion Between Numeric Types
- Unchecked Sign Extension
- Signed to Unsigned Conversion Error
- Unsigned to Signed Conversion Error
- Numeric Truncation Error
- Incorrect Calculations (142)
- Incorrect Calculation of Buffer Size
- Integer Overflow or Wraparound
- Integer Underflow (Wrap or Unwrap)
- Off-by-one Error
- Divide by Zero

Representation Errors

- Clearing, 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 Objects
- Discrepancy Information Leak
- Inter-Messages Information Leak (129)
- Cross-Boundary Channeling Information Leak
- Untrusted Information Leak
- Process Environment Information Leak
- Information Leak Through Debug Information
- Sensitive Information Unchecked Before Release
- Information Leak of System Data
- Information Leak Through Caching
- Information Leak Through Environment Variables
- File and Directory Information Leaks
- Information Leak Through Query Strings in GET Request
- Information Leak Through Logging of Private Data
- Information Loss or Distortion
- Containment Errors (Containment Error)

Improper Access of Indexable Resource (Range Error)

Type Errors

Improper Bounding or Escaping of Output (114)

String Errors

Data Structure Issues

Improper Handling of Syntactically Invalid Structure

Modification of Assumed-Immutable Data (MAID)

- Improper Input Validation (20)
- Pathname Traversal and Enumeration Errors
- Process Control
- Missing EOL Validation
- Failure to Sanitize Data into a Different Plane (Injection)
- Improper Sanitization of Special Elements used in a Context (Command Injection) (7)
- Failure to Preserve Web Page Structure (Close-able Scripting) (19)
- Improper Sanitization of Special Elements used in an SQL Command (SQL Injection) (76)
- Failure to Sanitize Data into LDAP Queries (LDAP Injection)
- XML Injection (aka Blind XML 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 Capitalize Remote Control Fields (Cookie Injection) (194)
- Improper Sanitization of Special Elements
- Technology Specific Input Validation Problems
- Misrepresentation of Input
- Unchecked Input for Loop Condition
- Bad Type Intersection Error (Panic Null Byte)
- Direct Use of Unsafe API
- Improper Output Sanitization for Logs
- Failure to Choose Separators within the Bounds of a Memory Buffer (113)
- 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 Extension Error
- Unvalidated Function Hook Arguments
- External Control of File Name or Path (124)
- Improper Address Validation in IOCTL with METHOD_IOCTL_IOCTL Control Code
- Use of Path Manipulation Function without Maximum and Buffer

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.codesecurify.org/Wiki/view.aspx/Security_Code_Reviews
- Security Code Review — Use Visual Studio Bookmarks To Capture Security Findings; <http://blogs.msdn.com/aliq/archive/2008/01/24/security-code-review-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

Testing

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

Fuzz testing

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

Resources

- Fuzz Testing of Application Reliability, University of Wisconsin; <http://pages.cs.wisc.edu/~bart/fuzz/fuzz.html>
- Automated Whitebox Fuzz Testing, Michael Levin, Patrice Godefroid and Dave Molnar, Microsoft Research; <ftp://ftp.research.microsoft.com/pub/tr/TR-2007-58.pdf>
- IANewsletter Spring 2007 "Look out! It's the 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/>



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

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)

MS08-078 and the SDL ★★★★★

Hi, Michael here.

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

The Common Vulnerabilities and Exposures (CVE) entry for this bug is [CVE-2008-1844](#).

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

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

Background

The bug was an invalid pointer dereference in MSHTMLDILL 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 `_anyPxfEr`, 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...

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](#).

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

TOCTOU issues. We will update our training to address this.

Our static analysis tools don't find this because the tools would need to understand the re-entrant nature of the code.

Fuzz Testing

OWASP Top Ten 2007 & 2010 use CWE refs



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

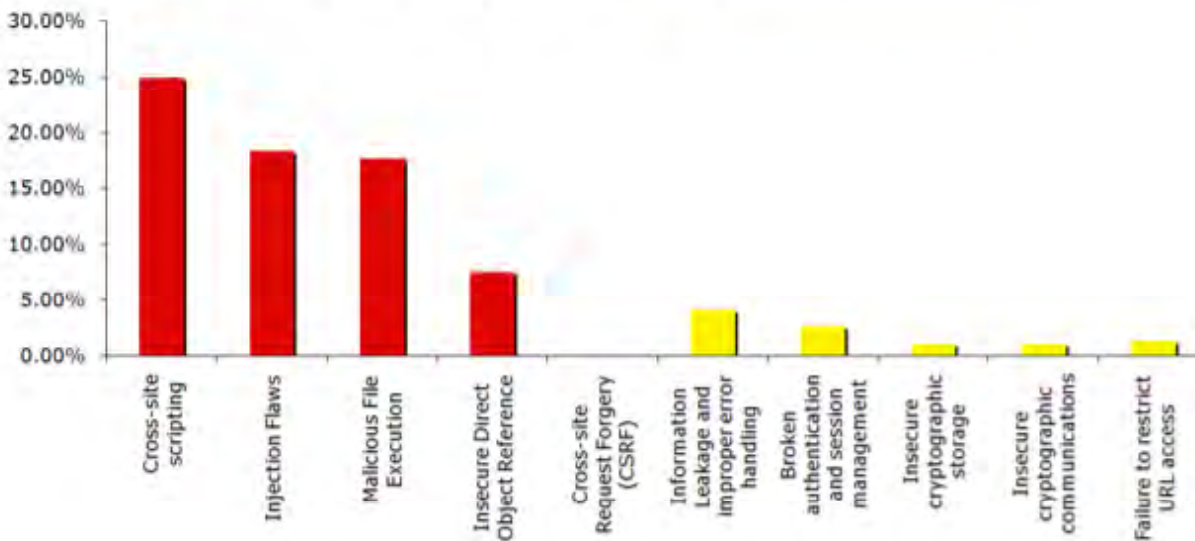


Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006



OWASP

The Open Web Application Security Project

 [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 [\[hide\]](#)

- 1 Introduction
 - 1.1 Why Does Code Have Vulnerabilities?
 - 1.2 What is Security Code Review?

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"

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

NVD XML feeds also include CWE

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

Resource Status

NVD contains:

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

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

Email List

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

- ☐ NVD Announcements
- ☐ SCAP Announcements
- ☐ SCAP Discussion List
- ☐ XCCDF Discussion List

Workload Index

[Vulnerability Workload Index](#): 9.06

About Us

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

Overview

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

Impact

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

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

References to Advisories, Solutions, and Tools

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

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

Vulnerable software and versions

Configuration 1
- Clansphere, Clansphere, 2007.4

Technical Details

Vulnerability Type (View All)
SQL Injection ([CWE-89](#))

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

Common Platform Enumeration:

Vulnerability Type (View All)
SQL Injection ([CWE-89](#))

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

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

CWE-89 Individual Dictionary Definition (Draft 9)

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

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

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

Likelihood of Exploit: Very High

Common Consequences:
Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.
Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.
Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability.
Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.
Requirements specification: A non-SQL style database which is not subject to this flaw may be chosen.
Design: Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data.
Design: Duplicate any filtering done on the client-side on the server side.
Implementation: Implement SQL strings using prepared statements that bind variables. Prepared statements that do not bind variables can be vulnerable to attack.

Potential Mitigations:

Section Contents

CWE List
Full Dictionary View
Classification Tree
Reports

Other Name of Interest
Sources

Key
W - Weakness
B - Base
V - Variant
F - Class
C - Chain
C - Composite
C - Category
V - View
D - Deprecated

WELCOME TO THE NIST SAMATE REFERENCE DATASET PROJECT

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

[Browse, download, and search the SRD](#)

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

[How to submit test cases](#)



Draft Special Publication 500-268

Source Code Security Analysis Tool Functional Specification Version 1.0

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

29 January, 2007

Michael Kass
Michael Koo

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

NIST Special Publications:

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

NIST Interagency Reports:

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



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

INL/EXT-10-10381

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

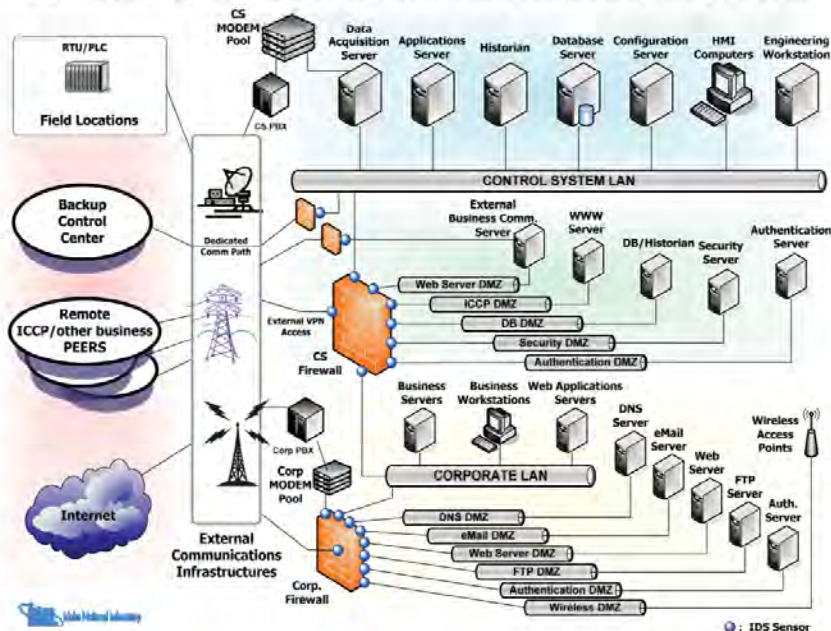
May 2010

NSTB

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



SECURE CONTROL SYSTEM/ENTERPRISE ARCHITECTURE



Idaho National Labs SCADA Report

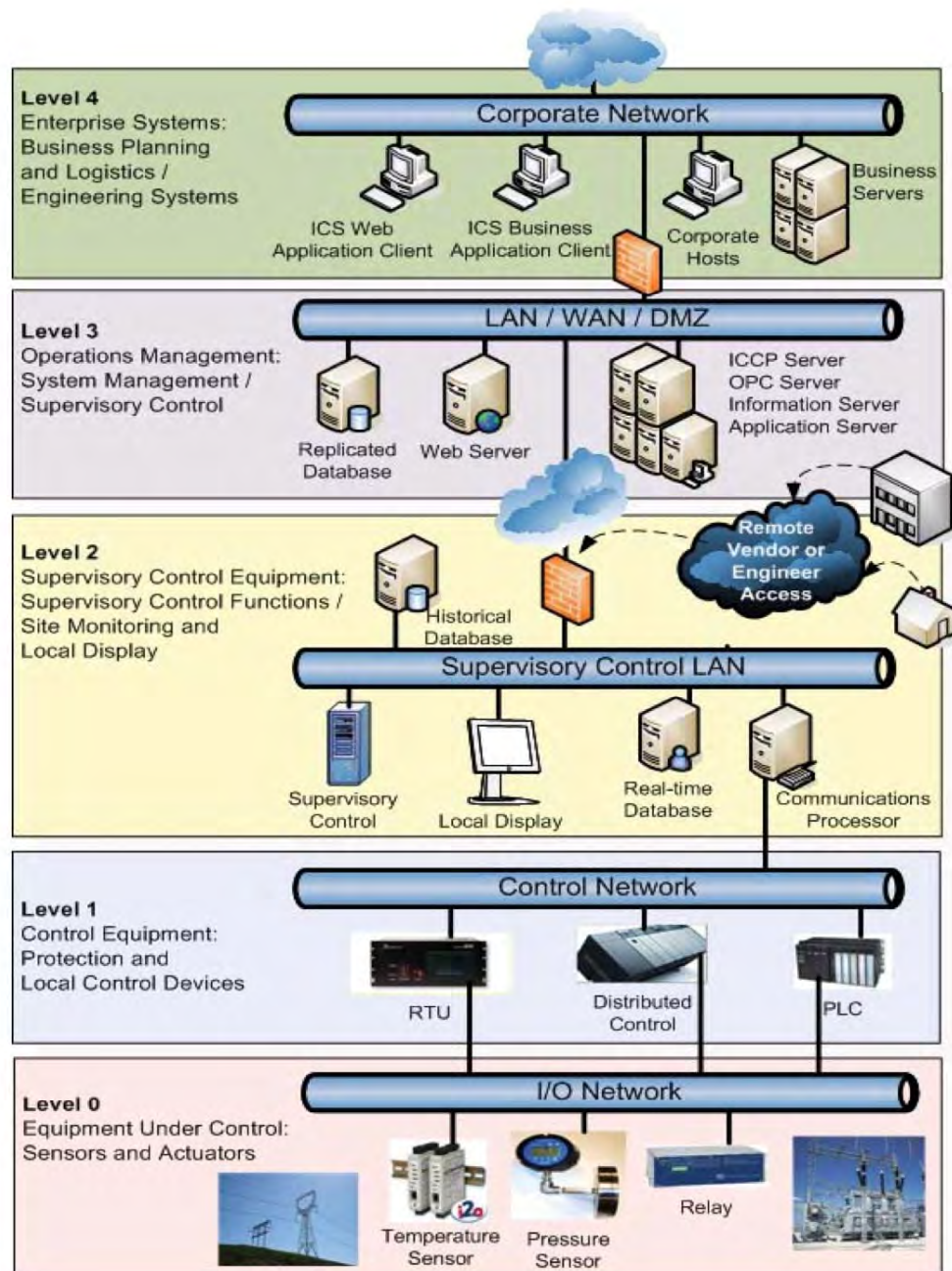


Table 27. Most common programming errors found in ICS code.

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-398: Indicator of Poor Code Quality	CWE-680: Integer Overflow to Buffer Overflow
	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-442: Web Problems	CWE-690: Unchecked Return Value to NULL Pointer Dereference
	CWE-772: Missing Release of Resource after Effective Lifetime
	CWE-22: Improper Limitation of a Pathname to a Restricted Directory (“Path Traversal”)
CWE-703: Failure to Handle Exceptional Conditions	CWE-79: Failure to Preserve Web Page Structure (“Cross-site Scripting”)
	CWE-89: Failure to Preserve SQL Query Structure (“SQL Injection”)
	CWE-431: Missing Handler
	CWE-248: Uncaught Exception
	CWE-755: Improper Handling of Exceptional Conditions
	CWE-390: Detection of Error Condition Without Action

Top 25 Series – Summary and Links

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

Filed under Top25

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

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

Pat on Some Thoughts About Passwords

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

Nick Owen on Some Thoughts About Passwords

ARCHIVES

Select Month

META

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



The Security Development Lifecycle

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

SEARCH

• [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 written coverage of the Top 25 and believe that the results tell 25 were developed independently of the software analysis white paper and guidance around every made many of the same for you to download and

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

CWE Title

20 Improper Input Validation
116 Improper Encoding or Escaping of Output

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

CWE Outreach: A Team Sport

May/June Issue of IEEE Security & Privacy...

CWE-732: Insecure Permission Assignment for Critical Resource

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

CWE-330: Use of Insufficiently Random Values

Identify all the random number generators in your code and determine which, if any, generate passwords, or other secret data. Make sure the code generating random numbers is cryptographically random and not a deterministic pseudorandom generator. The C runtime `rand()` function uses functions like `rand()` and `rand()` but not for cryptography.

CWE-250: Execution with Unnecessary Privileges

Identify all processes that run part of your solution and determine what privileges they require to operate correctly. If a process runs as root (on Linux, Unix, Mac OS X) or system (Windows), ask yourself, "Why?" Sometimes the answer is totally valid because the code must perform a privileged operation, but sometimes you don't know why it runs away other than, "That's the way it's always run!" If the code needs to operate at high privilege, keep the time span within which the code is high privilege as small as possible—for example, opening a port below 1024 in a LAN application requires the code be run as root, but after that,

Basic Training

portant that the file and path do not exist before you access a file or path. For example, if you are looking for a file or directory, or accessing a file or directory, make sure the name is appropriate to valid data. If you are using a "known good" way to access a file or directory, you can use the `File.Exists` method to verify the file exists before you attempt to access it.

CWE-428: Untrusted Path

Old versions of Windows used the `File.Exists` method to verify the file exists before you attempt to access it. However, if the file had a weakly defined name, the `File.Exists` method would not guarantee the file existed. Searches or searches on a path, but this is not a good practice. For example, if you are using a "known good" way to access a file or directory, you can use the `File.Exists` method to verify the file exists before you attempt to access it.

CWE-94: Failure to Generate Unique Values

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

(XSS). CWE-79 is the real bug that makes CWE-116 worse. In the past, we took XSS bugs lightly, but now we see worms that can exploit XSS vulnerabilities in social networks such as MySpace (for example, the Smurf 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.

CWE-119: Failure to Constrain Memory Operations

The dreaded buffer overflow is a scourge of C and C++ and a vulnerability type that has more headaches than buffer runs. The best way to reduce the problem is to move away from C and C++ where it makes sense and use higher-level languages such as Ruby, C#, and so on, because they don't offer direct access to memory. For C and C++ applications, developers should use "known bad" functions such as `strcpy`, `strcpy_s`, `strcat`, `strcat_s`, `strcat_s`, and so on, and use secure versions. Visual C++ has many weak APIs at compile time and you should strive to use the secure versions. Also, fuzz testing and static analysis can help identify potential buffer overruns. Operating-system-level solutions such as address space layout randomization and no-execution can help reduce the chance of a buffer overrun is exploited.

CWE-642: External Control of Critical State Data

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

CWE-73: External Control of Filename or Path

Attackers might be able to control arbitrary file data if they can control the data that's used as part of a path name. It's critical

Basic Training

Editors: Richard Ford, rford@se.sri.edu
Michael Howard, mhoward@microsoft.com

Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

In January 2009, MITRE and SANS issued the "2009 CWE/SANS Top 25 Most Dangerous Programming Errors" to help make developers more aware of the bugs that can cause security compromises (<http://cwe.mitre.org/top25>). I was one of the many people

MICHAEL HOWARD
Microsoft

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

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

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

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

CWE-20: Improper Input Validation

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

CWE-116: Improper Output Encoding

You could really isn't

Basic Training

68 Improving Software Security by Eliminating the CWE Top 25 Vulnerabilities

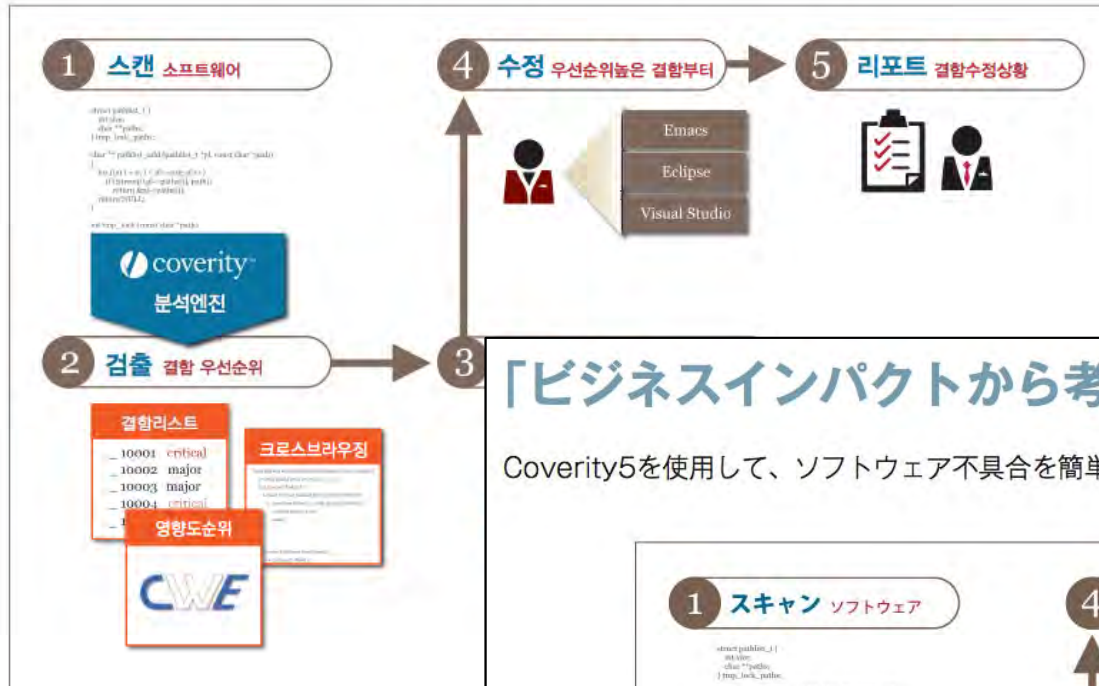
MICHAEL HOWARD





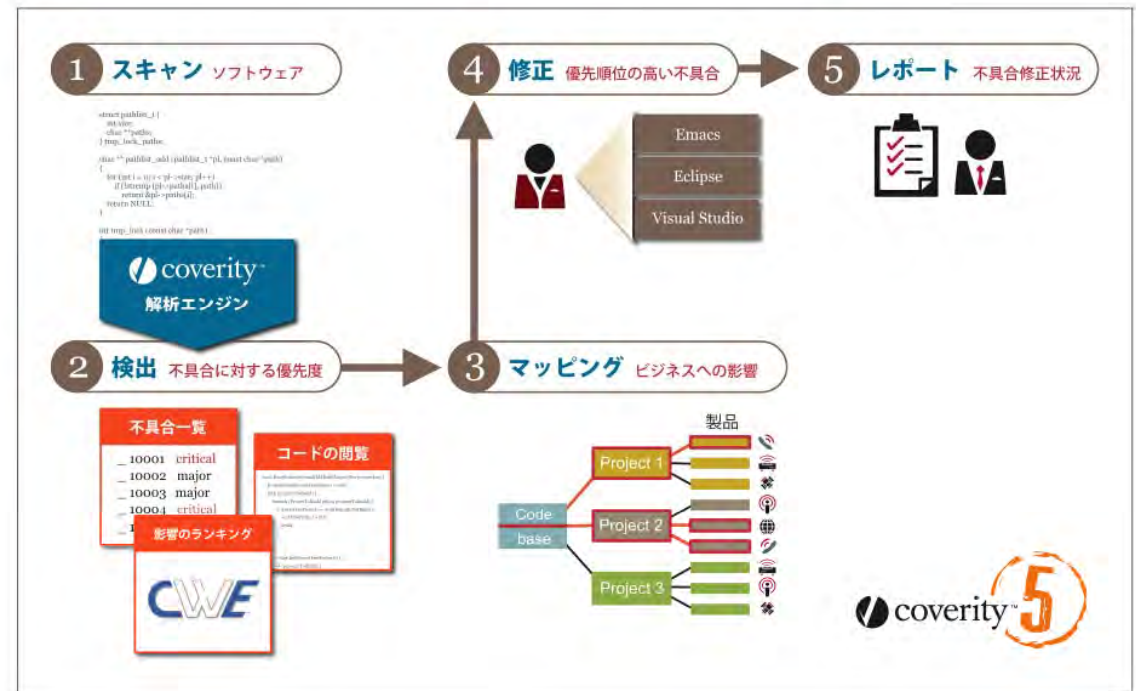
[비즈니스 임팩트를 줄여주는 새로운 품질 관리 방법론]

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」

Coverity5를 사용하여, 소프트웨어 불합을 간단히 제거하는 5스텝은以下の通りです.



Korean

Making Security Measurable™ **Japanese**




Threat Classification Taxonomy Cross Reference View

Page history

✔ Check for plagiarism

Threat Classification 'Taxonomy Cross Reference View'

SideBar 

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

WASC Projects

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?qid=83336>

Recent Activity

-  **Insufficient Data Protection Working**
edited by Robert Auger

Resource

DoD Information Assurance
Certification and Accreditation
Process (DIACAP)

The DIACAP defines the minimum standards for information systems that are accredited by the DoD and authorized to process information. It defines application-level security controls, but it is not a framework for security activities, general tasks, and a management process.

Defense Information Systems
Agency (DISA)

The DISA provides a security technical in development that offer more granular information and vulnerability assessment techniques. The check

U.S. Department of Homeland
Security (DHS)

The DHS offers information on security best practices and tools for application- and software-related security, as 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 vulnerabilities, how to identify, fix and avoid them.

Digital Building Security In Maturity Model (BSIMM)

Created by Cigital, an IBM Business Partner, the BSIMM is designed to help organizations 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 globe. The IBM X-Force team is an excellent resource for trend analysis and answers to questions about attacks that are most common, where they are coming from and what organizations can do to reduce the risks.

IBM Institute for Advanced
Security (IAS)

This companywide cybersecurity initiative applies IBM research, services, software and tools to help governments and other clients improve the security and resiliency of their IT and business operations.

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 Enumeration⁵ (CWE) list and the Common

Vulnerability E
be tested. Thi
information an
and vulnerabi
against the m

Creating a se
plan includes

⁵ For more inform

⁶ For more inform

10 Security in Development:

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

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
Unlimited distribution subject to the copyright.

<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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CSIRW '09, April 13-15, Oak Ridge, Tennessee, USA
Copyright © 2009 ACM 978-1-60558-518-5 ... \$5.00

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

Making
Security
Measurable™

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.

Copyright © 2010 T. J. Halloran

This material is based upon work supported by the following grants: NASA, NCC2-1298 and NNA00C530A; Lockheed Martin: RRNH51798; ARD, DAAD190210380; 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 Defense Guidelines

What the 20 CSC Critics say...

20 Critical Security Controls - Version 2.0

- 20 Critical Security Controls - Introduction (Version 2.0)
- Critical Control 1: Inventory of Authorized and Unauthorized Devices
- Critical Control 2: Inventory of Authorized and Unauthorized Users
- Critical Control 3: Secure Configurations for Hardware and Software
- Critical Control 4: Secure Configurations for Network Devices
- Critical Control 5: Boundary Defense
- Critical Control 6: Maintenance, Monitoring, and Analysis of Security
- Critical Control 7: Application Software Security
- Critical Control 8: Controlled Use of Administrative Privileges
- Critical Control 9: Controlled Access Based on Need to Know
- Critical Control 10: Data Protection
- Critical Control 11: Incident Response and Reporting
- Critical Control 12: Business Continuity and Disaster Recovery
- Critical Control 13: Security Awareness and Training
- Critical Control 14: Vendor Management
- Critical Control 15: Physical Security
- Critical Control 16: Security Testing
- Critical Control 17: Security Assessments
- Critical Control 18: Security Monitoring
- Critical Control 19: Security Incident Response
- Critical Control 20: Security Program Management

Making
Security
Measurable™

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 the

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.



Linkage with Fundamental Changes in Enterprise Security Initiatives

Enabling Distributed Security in Cyberspace

Building a Healthy and Resilient Cyber Ecosystem with Automated Collective Action

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

- **Technical Interoperability.** The ability for different technologies to communicate and exchange data based upon well defined and widely adopted interface standards.
- **Policy Interoperability.** Common business processes related to the transmission, receipt, and acceptance of data among participants.

Within cybersecurity, all three types of interoperability are being enabled through an approach that has been refined over the past decade by many in industry, academia, and government. It is an information-oriented approach, generally referred to as [cyber] security content automation and comprises the following elements.¹³

- **Enumerations.** These are lists or catalogs of the fundamental entities of cybersecurity, for example, cyber devices and software items (CPE); device and software

businesses in architecture, design, or code (CWE); or publicly known attack patterns (CAPEC).

ability. Credit card in leverages scores for this severity level. te enumerations and support the creation of assessment results, audit logs, messages, associated with assets, configurations, DF & OVAL); security announcements (CAIF), associated with vulnerability (CVSS), sensor ERT security bulletins and incident reports technical interoperability.

broad collection of best practices, s, checklists, tools, guidelines, rules, and s, knowledge repositories serve as the enable policy interoperability. Examples housed on the National Checklist Program rtment of Defense Security Technical or guides."

supported security content automation efforts . Projections are based on current resourcing irected community. Figure 4 also illustrates functionality over time (e.g., the expansion of tops to networks).

e of the various named standards.

March 23, 2011

12

March 23, 2011

us-cert.gov https://buildsecurityin.us-cert.gov/swa/forums.html

Software Assurance

Community Resources and Information Clearinghouse

Sponsored by DHS National Cyber Security Division

Search GO 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

Forums

The Software Assurance Program of the Department of Homeland Security Cyber Security Division co-sponsors SwA Forums semi-annually with the Department of Defense and the National Institute for Standards and Technology. The purpose of the forums is to bring together members of government, industry, academia with vested interests in software assurance to discuss and promote security, and reliability in software.

FORUM PRESENTATIONS

SwA Forum presentations that are released for publication are posted here:

13th Semi-Annual Software Assurance Forum - September 27-October 1, 2010

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

In between SwA Forums, the DHS SwA Program hosts SwA Working Groups. These groups provide venues for public-private collaboration in advancing software assurance initiatives, and status updates from the SwA Working Groups are presented at the Forums and to other relevant stakeholder groups. For more information on WG sessions, see the Events page on Build Security In.

- June 21-23, 2010 Working Group Session Agenda and Presentations
- December 14-16, 2010 Working Group Session Agenda and Presentations

Learn more about SwA Forums and Working Group Sessions or download our Working Group Sessions Fact Sheet and Frequently Asked Questions.



Homeland Security

Software Assurance Technology, Tools and Product Evaluation Working Group Resources

us-cert.gov https://buildsecurityin.us-cert.gov/swa/ttperesrc.html

GO Google

Software Assurance

Community Resources and Information Clearinghouse

Sponsored by DHS National Cyber Security Division

Search GO 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](#)

We speak CVE
cve.mitre.org

We speak CWE
cwe.mitre.org

We use OVAL

We know MAEC
maec.mitre.org

U.S. CERT
United States Computer Emergency Response Team

Homeland Security



Contact

Technical comments or questions:
cve@mitre.org
Phone: 1-800-441-2391

General questions or suggestions:
cwe@mitre.org
Phone: 1-800-441-2391
maec@mitre.org
Phone: 1-800-441-2391

Information available at: <http://www.us-cert.gov>



Questions?

ramartin@mitre.org