Homeland Security

Commerce

National Defense

BUILDING SECURITY IN

SOFTWARE ASSURANCE

Uptake of CWE and CAPEC

http://cwe.mitre.org

CWE™

CAPEC™

http://capec.mitre.org

Robert A. Martin

3 March 2011

MITRE

CWE web site visitors by City

CWE Compatibility & Effectiveness Program

(launched Feb 2007)

CWE – CWE Compatibility

http://cwe.mitre.org/compatible/index.html

Common Weakness Enumeration

A community-developed dictionary of common software weaknesses

Home > Compatibility | View the CWE List

CWE List
Full Dictionary View

CWE Compatibility

Section Contents
Compatibility

Organizations Participating

cwe.mitre.org/compatible/

All organizations participating in the CWE Compatibility and Effectiveness Program are listed below, including those with CWE-Compatible Products and Services and those with Declarations to Be CWE-Compatible.

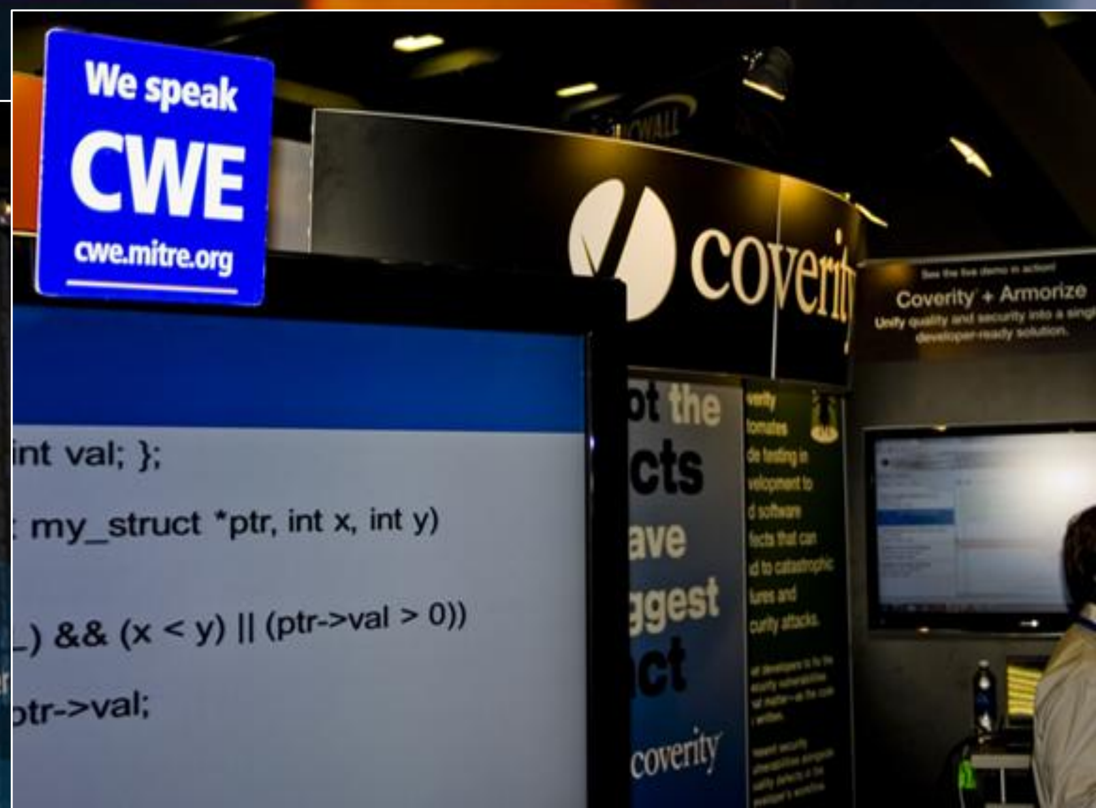
TOTALS
Organizations Participating: **29**
Products & Services: **48**

December 29, 2006

Products are listed alphabetically by organization name:


armorize
We speak
CWE
cwe.mitre.org
The Web Malware Experts


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;

CWE Coverage – Implemented…

Coverity Data Sheet

coverity

Coverity Coverage for Common Weakness Enumeration (CWE): Java

CWE ID	Coverity Static Analysis Checker
171	BAD_EQ
252	CHECKED_RETURN
366	GUARDED_BY_VIOLATION
	INDIRECT_GUARDED_BY_VIOLATION
	NON_STATIC_GUARDING_STATIC
	VOLATILE_ATOMICITY
382	DC.CODING_STYLE
	BAD_OVERRIDE
	DC.EXPLICIT_DEPRECATION
	DC.GC
	MUTABLE_COMPARISON
398	MUTABLE_HASHCODE

Coverity Data Sheet

coverity

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

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

CWE IDs mapped to Klocwork Java issue types - current http://www.klocwork.com/products/documentation/curren...

CWE IDs mapped to Klocwork Java issue types

From current

CWE IDs mapped to Klocwork Java issue types

See also Detected Java Issues.

...escription
...goes to native code
...tampering ...ction
...Working Directory
(Stored XSS)
...g (Reflected XSS)
(Stored XSS) ...g (Reflected XSS)
...d information from the ...ents
...orms: validate method
...orms: inconsistent validate
...e Splitting
...ex used for array access

2/26/11 10:35 AM

CWE IDs mapped to Klocwork C and C++ issue types/ja -... http://www.klocwork.com/products/documentation/curren...

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

From current

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

その他の情報 Detected C and C++ Issues.

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

1 of 7 2/26/11 10:34 AM

CENZIC www.cenzic.com | (866) 4-CENZIC (866-423-6942)

Cenzic Product Suite is CWE Compatible

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

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

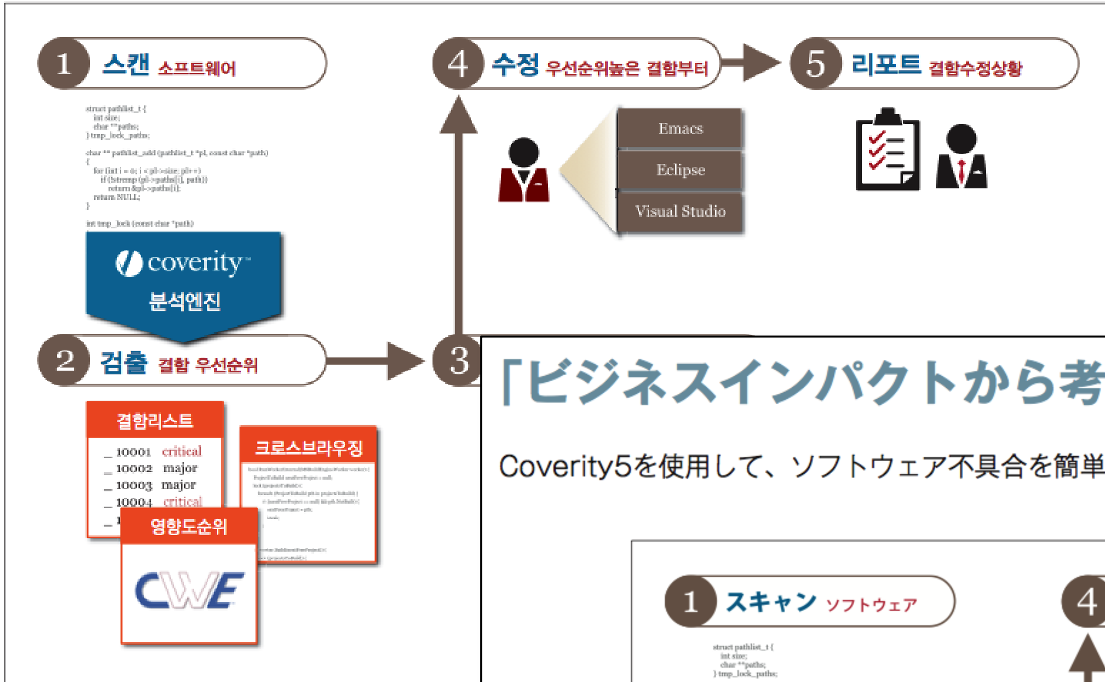
The following is a mapping between Cenzic's SmartAttacks and CWE ID's:

	Cenzic SmartAttack Name	CWE ID/s
1	Application Exception	**CWE-388:** Error Handling
2	Application Exception (WS)	**CWE-388:** Error Handling
3	Application Path Disclosure	**CWE-200:** Information Leak (rough match)
4	Authentication Bypass	**CWE-89:** Failure to Sanitize Data into SQL Queries (aka 'SQL Injection') (rough match)
5	Authorization Boundary	**CWE-285:** Missing or Inconsistent Access Control, CWE-425: Direct Request ('Forced Browsing')
6	Blind SQL Injection	**CWE-89:** Failure to Sanitize Data into SQL Queries (aka 'SQL Injection')
7	Blind SQL Injection (WS)	**CWE-89:** Failure to Sanitize Data into SQL Queries (aka 'SQL Injection')
8	Browse HTTP from HTTPS List	**CWE-200:** Information Leak
9	Brute Force Login	**CWE-521:** Weak Password Requirements
10	Buffer Overflow	**CWE-120:** Unbounded Transfer ('Classic Buffer Overflow')
11	Buffer Overflow (WS)	**CWE-120:** Unbounded Transfer ('Classic Buffer Overflow')
12	Check Basic Auth over HTTP	**CWE-200:** Information Leak
13	Check HTTP Methods	**CWE-650:** Trusting HTTP Permission Methods on the Server Side

Cenzic CWE Brochure | October 2009 1

[비즈니스 임팩트를 줄여주는 새로운 품질 관리 방법론]

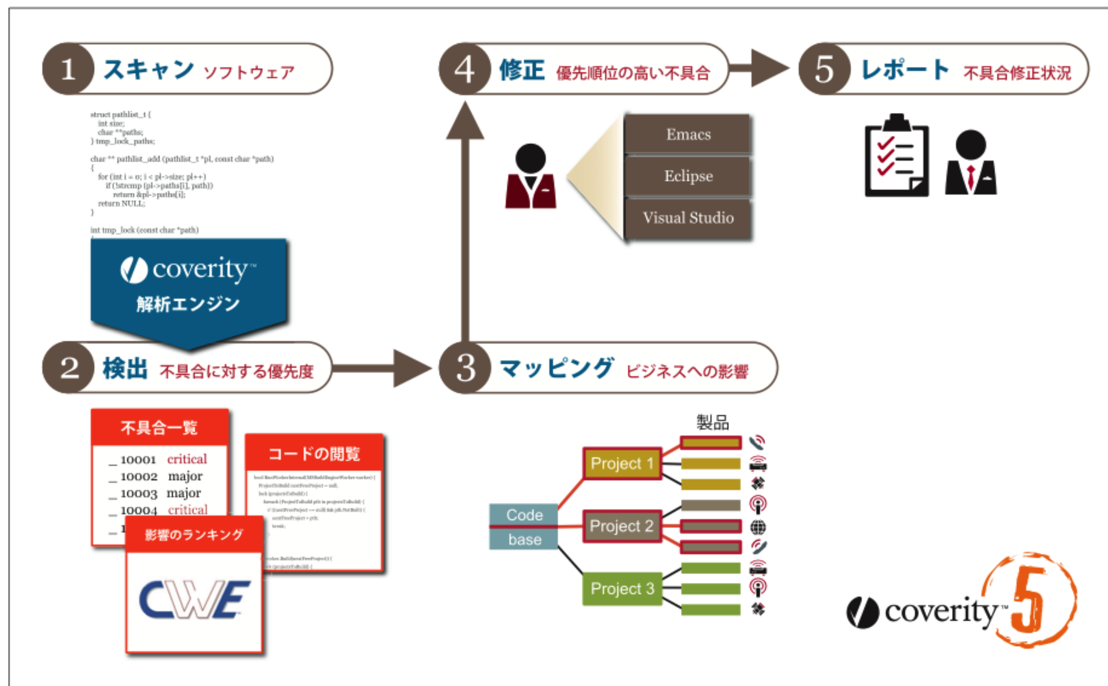
y5를 사용하여, 소프트웨어 결함을 없애는 5가지 스텝은 아래와 같습니다.



Korean

「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。



Japanese

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.

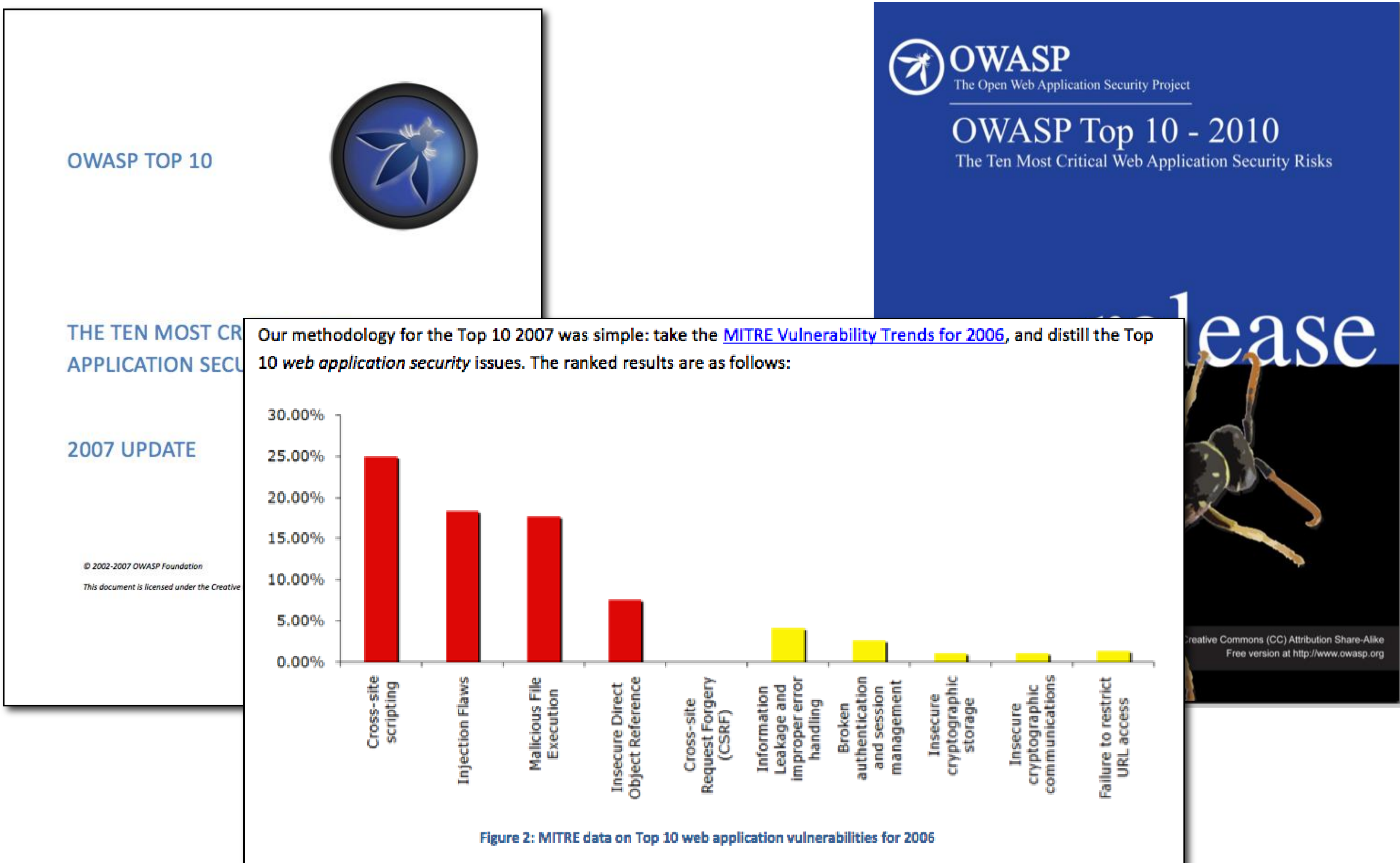
An Overview of the Steps:

(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

OWASP Top Ten 2007 & 2010 use CWE refs



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"

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

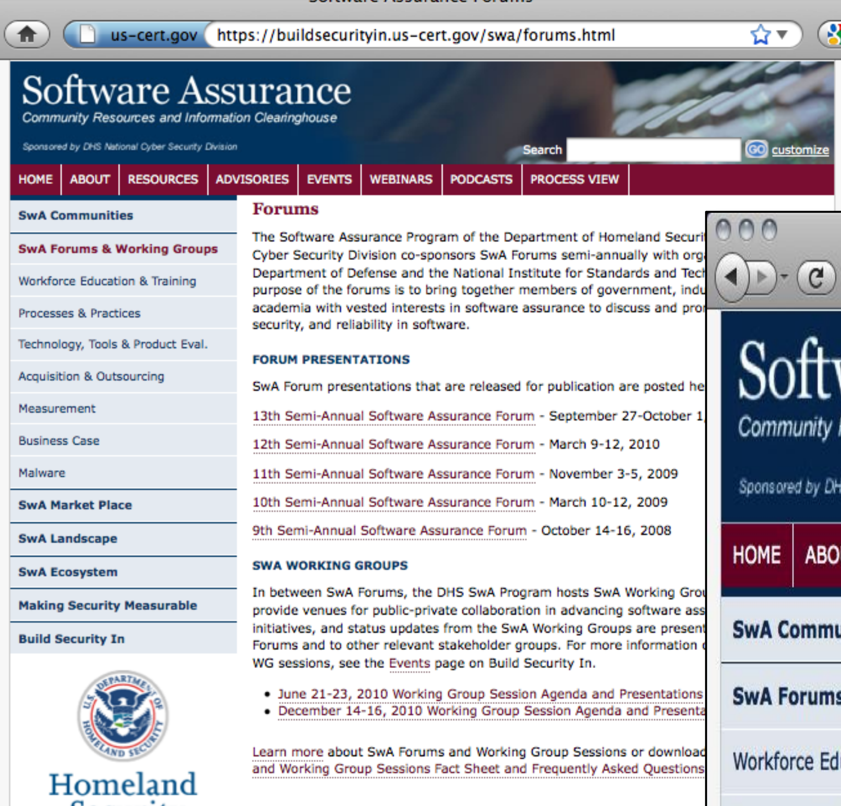
- http://twitter.com/wascupdates

Join us on Linkedin!

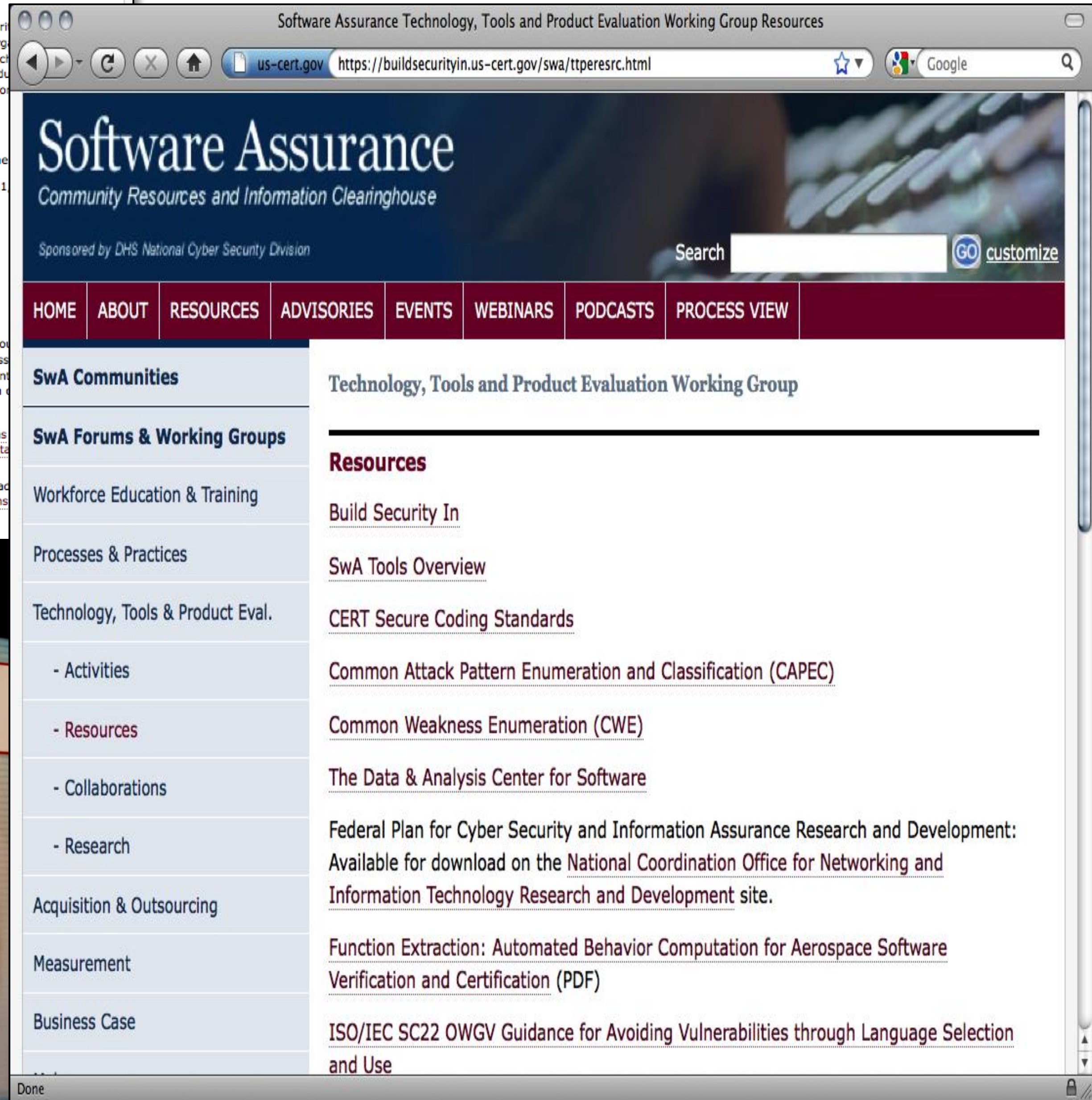
- http://www.linkedin.com/groups?gid=83336

Recent Activity

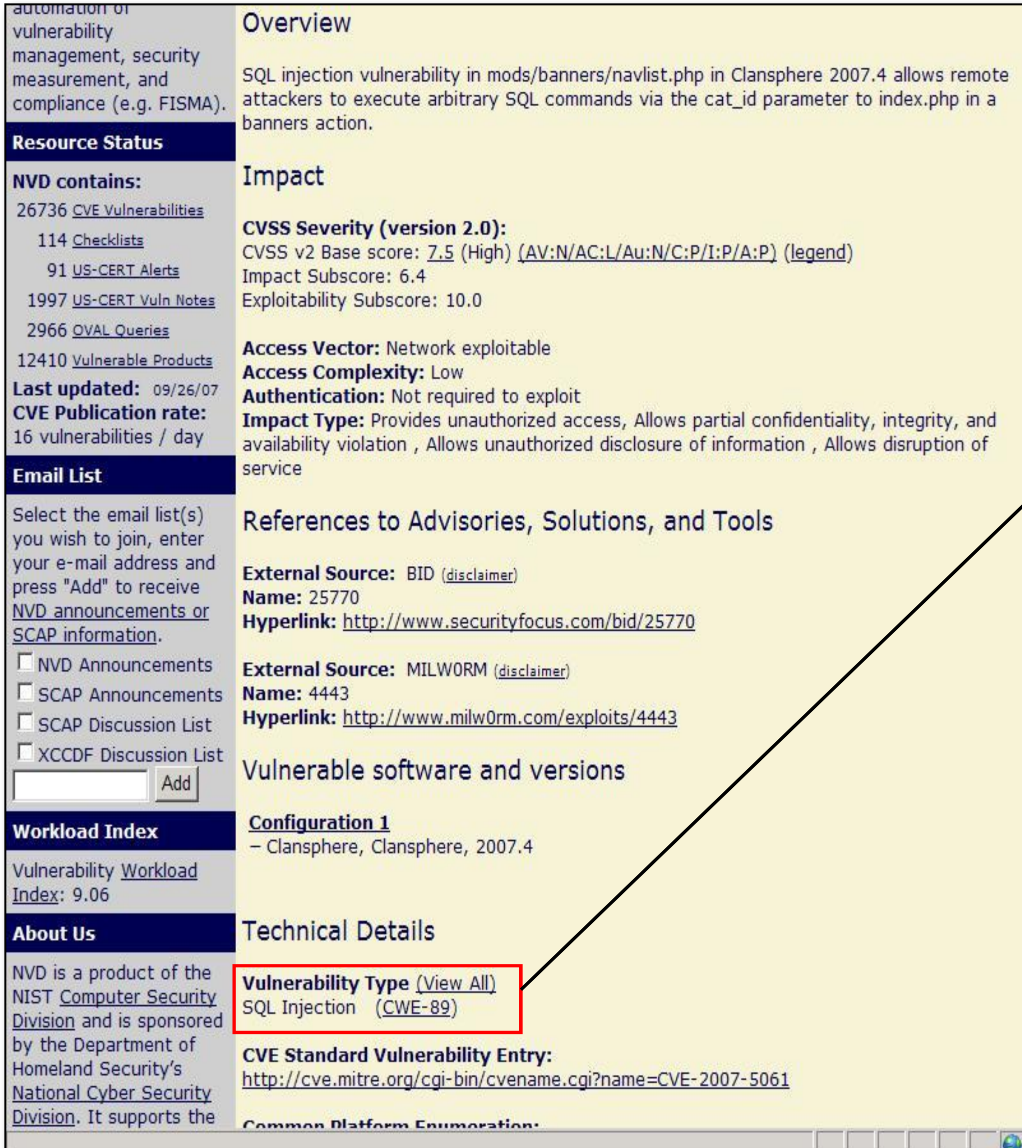
Insufficient Data Protection Working
edited by Robert Auger


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
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
The Software Assurance Program of the Department of Homeland Securi
Cyber Security Division co-sponsors SwA Forums semi-annually with org
Department of Defense and the National Institute for Standards and Tech
purpose of the forums is to bring together members of government, indu
academia with vested interests in software assurance to discuss and pro
security, and reliability in software.
FORUM PRESENTATIONS
SwA Forum presentations that are released for publication are posted he
13th Semi-Annual Software Assurance Forum - September 27-October 1
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 Grou
provide venues for public-private collaboration in advancing software ass
initiatives, and status updates from the SwA Working Groups are present
Forums and to other relevant stakeholder groups. For more information
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 Presenta
Learn more about SwA Forums and Working Group Sessions or download
and Working Group Sessions Fact Sheet and Frequently Asked Questions


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
Contact


Software Assurance Technology, Tools and Product Evaluation Working Group Resources
us-cert.gov https://buildsecurityin.us-cert.gov/swa/ttperesrc.html
Google
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

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



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

Resource Status

NVD contains:

26736 CVE Vulnerabilities
114 Checklists
91 US-CERT Alerts
1997 US-CERT Vuln Notes
2966 OVAL Queries
12410 Vulnerable Products

Last updated: 09/26/07

CVE Publication rate: 16 vulnerabilities / day

Email List

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

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

Add

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: MILW0RM (disclaimer)
Name: 4443
Hyperlink: http://www.milw0rm.com/exploits/4443

Vulnerable software and versions

Configuration 1
– Clansphere, Clansphere, 2007.4

Technical Details

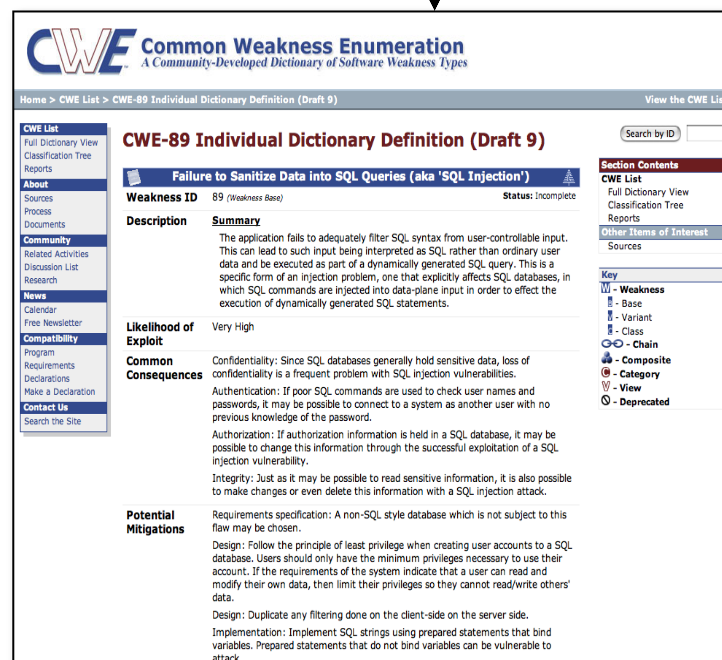
Vulnerability Type (View All)
SQL Injection (CWE-89)

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

Common Platform Enumeration:

NVD XML feeds also include CWE

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



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 List: Full Dictionary View, Classification Tree, Reports
About: Sources, Process, Documents
Community: Related Activities, Discussion List, Research
News: Calendar, Free Newsletter
Compatibility: Program, Requirements, Declarations, Make a Declaration
Contact Us: Search the Site

CWE-89 Individual Dictionary Definition (Draft 9)

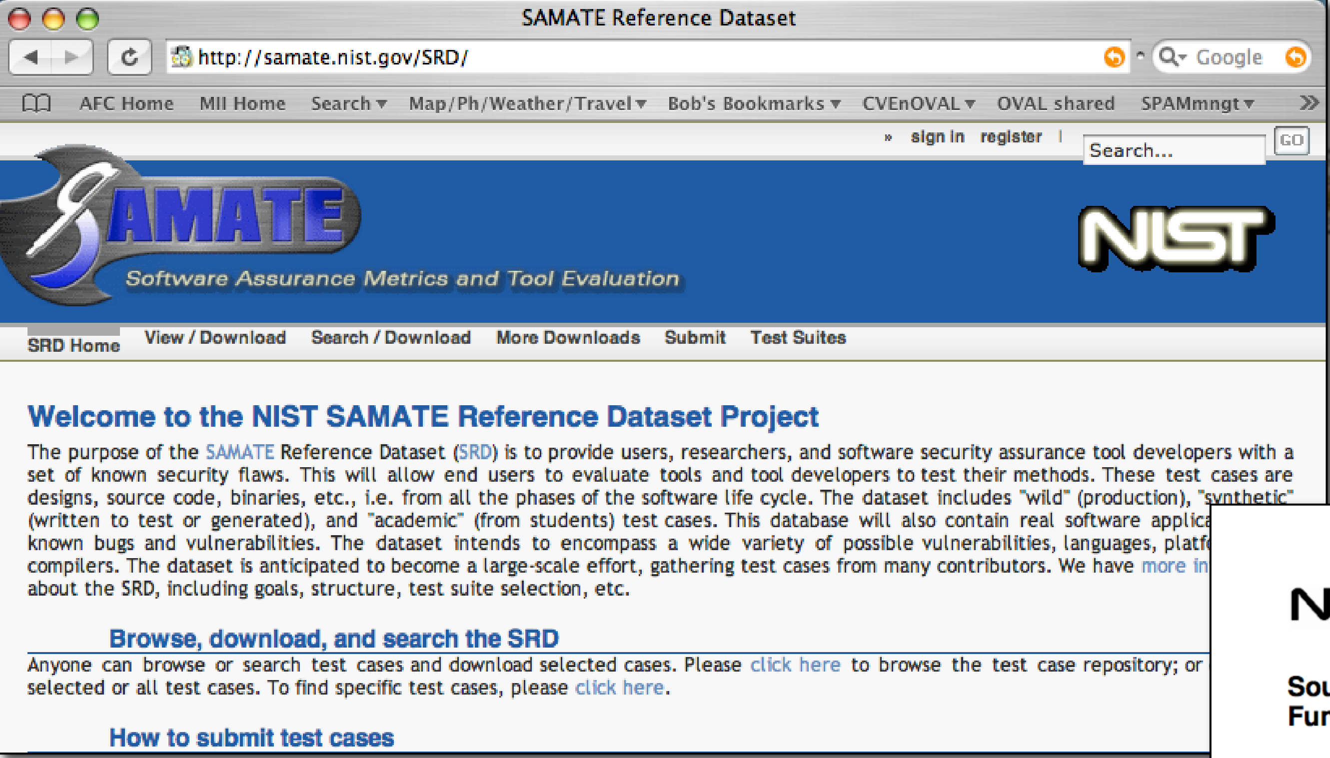
Search by ID

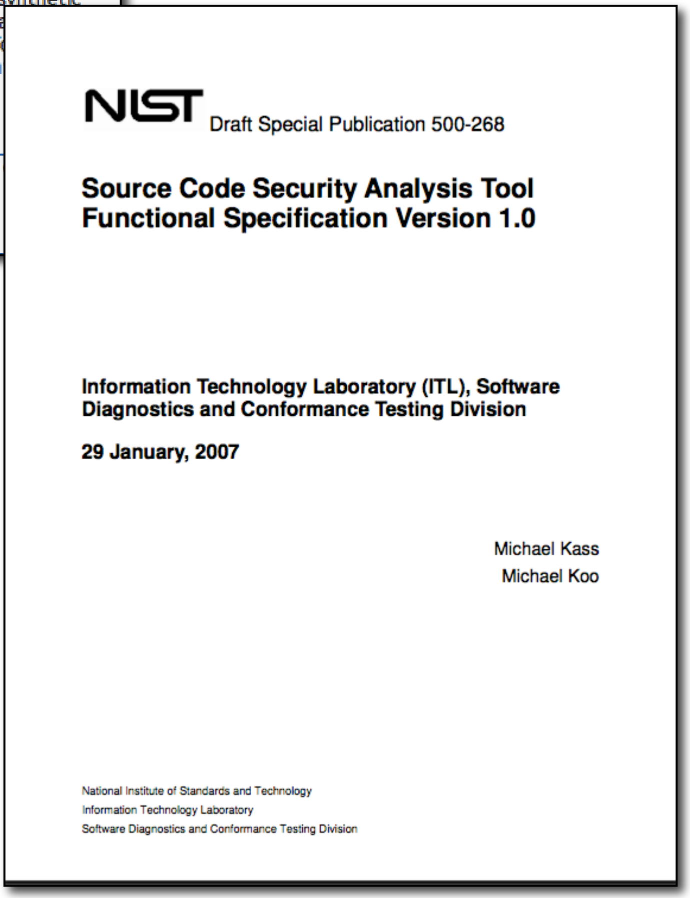
Failure to Sanitize Data into SQL Queries (aka 'SQL Injection')

Weakness ID	89 *(Weakness Base)* — **Status:** Incomplete
Description	**Summary** The application fails to adequately filter SQL syntax from user-controllable input. This can lead to such input being interpreted as SQL rather than ordinary user data and be executed as part of a dynamically generated SQL query. This is a specific form of an injection problem, one that explicitly affects SQL databases, in which SQL commands are injected into data-plane input in order to effect the execution of dynamically generated SQL statements.
Likelihood of Exploit	Very High
Common Consequences	Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities. Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password. Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability. Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.
Potential Mitigations	Requirements specification: A non-SQL style database which is not subject to this flaw may be chosen. Design: Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data. 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.

Section Contents
CWE List: Full Dictionary View, Classification Tree, Reports
Other Items of Interest: Sources

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




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

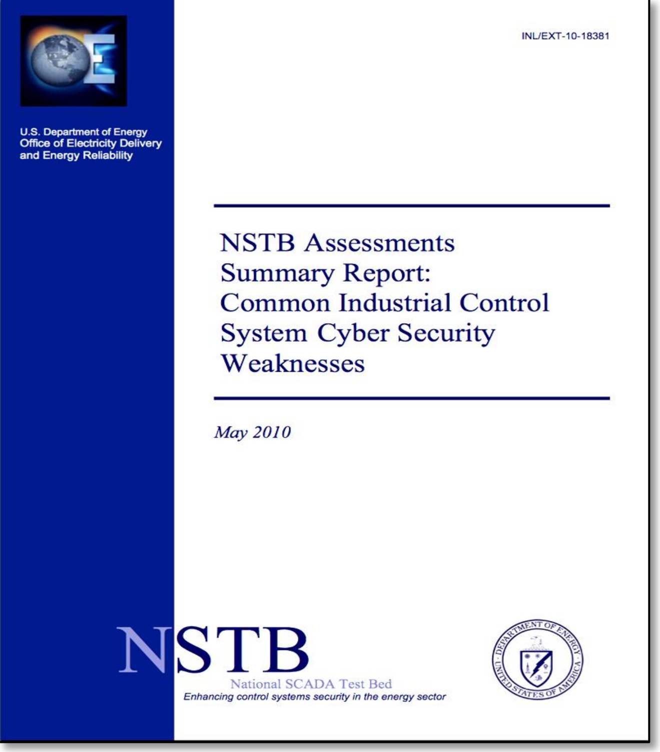
NIST Special Publications:

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

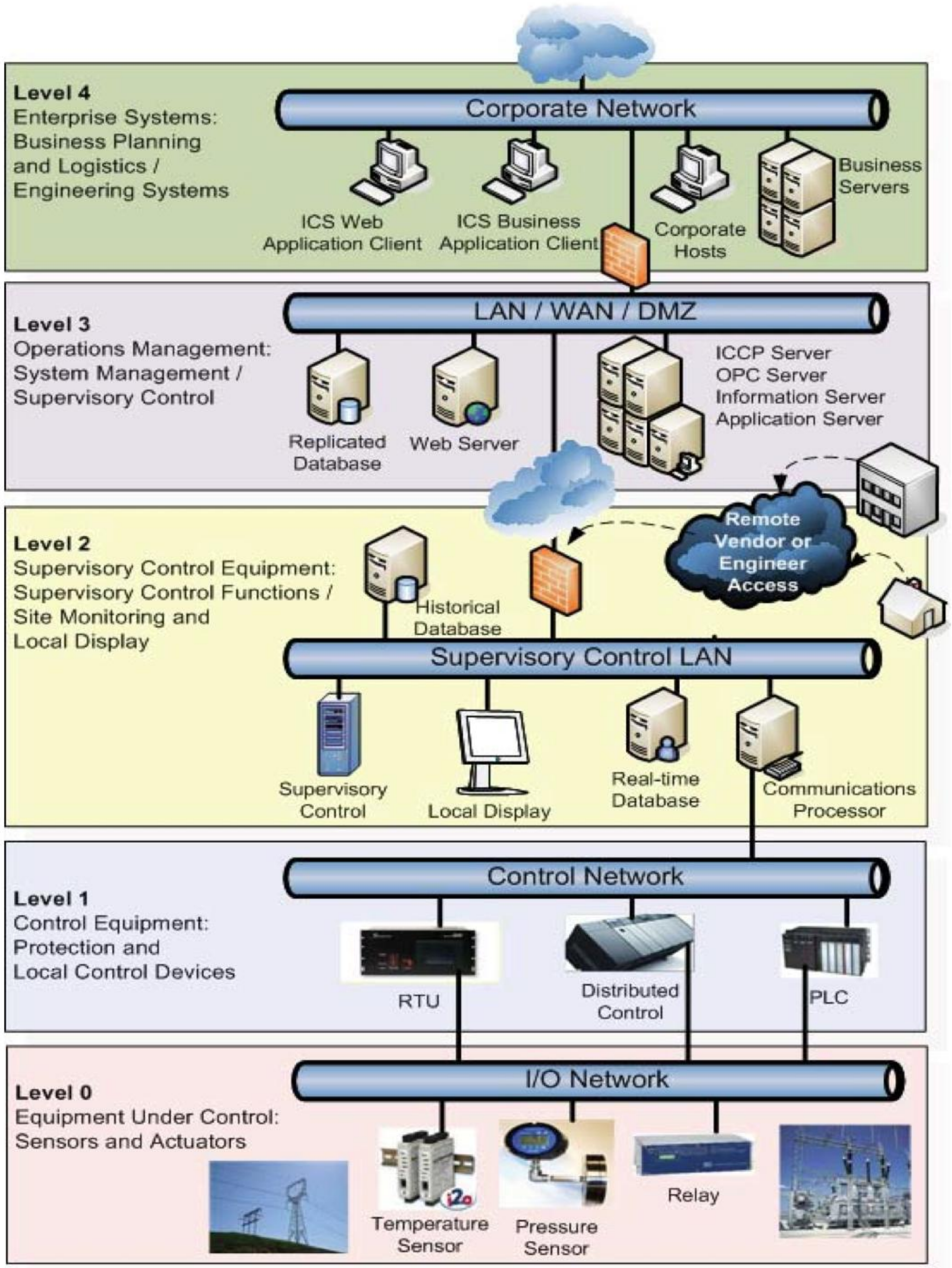

INL/EXT-10-18381

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

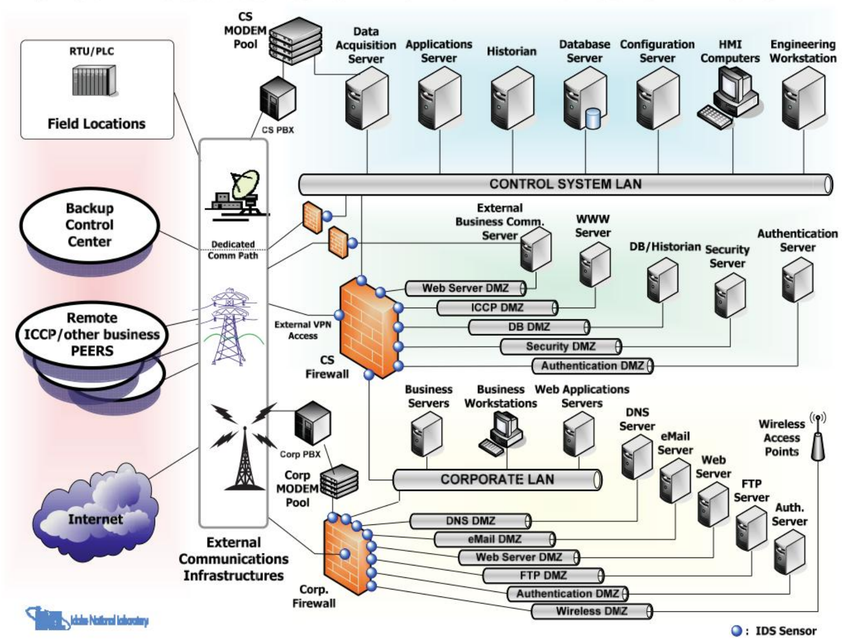


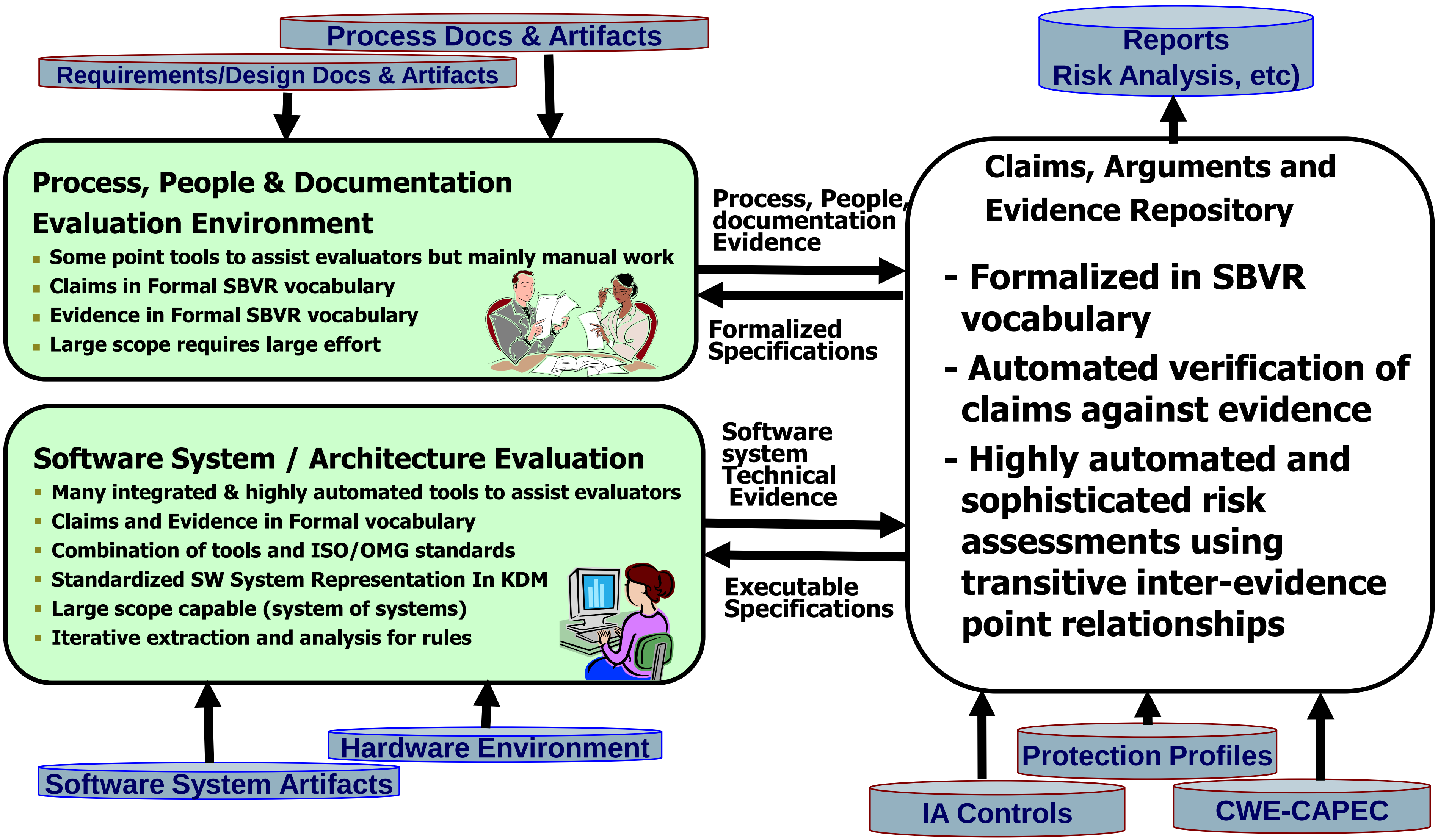
Table 27. Most common programming errors found in ICS code.

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

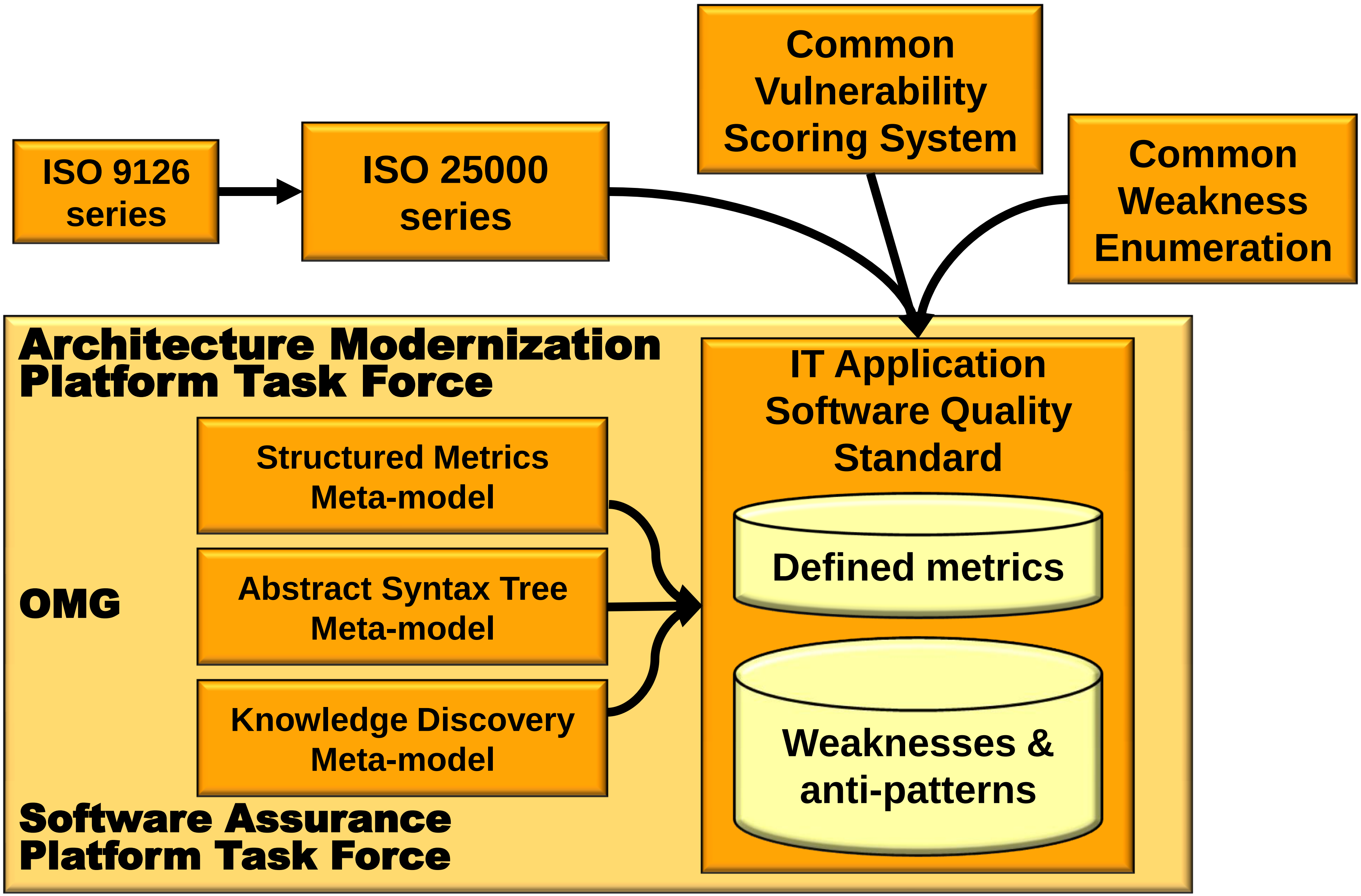
Software Assurance Ecosystem: The Formal Framework

OBJECT MANAGEMENT GROUP

The value of formalization extends beyond software systems to include related software system process, people and documentation



CISQ Standards Infrastructure



ISO/IEC JTC 1/SC 27 Nxxxx

ISO/IEC JTC 1/SC 27/WG x Nxxxxx

REPLACES: N

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

DOC TYPE:	NB NWI Proposal for a technical report (TR)
TITLE:	**National Body New Work Item Proposal on "Secure software development and evaluation under ISO/IEC 15408 and ISO/IEC 18405"**
SOURCE:	INCITS/CS1, National Body of (US)
DATE:	2009-09-30
PROJECT:	**15408 and 18405**
STATUS:	This document is circulated for consideration at the forthcoming meeting of SC 27/WG 3 to be held in Redmond (WA, USA) on 2nd – 6th November 2009.
ACTION ID:	ACT
DUE DATE:	
DISTRIBUTION:	P-, O- and L-Members W. Fumy, SC 27 Chairman M. De Soete, SC 27 Vice-Chair E. J. Humphreys, K. Naemura, M. Bañôn, M.-C. Kang, K. Rannenberg, WG-Conveners
MEDIUM:	Livelink-server
NO. OF PAGES:	xx

Secretariat ISO/IEC JTC 1/SC 27 –
DIN Deutsches Institut für Normung e. V., Burggrafenstr. 6, 10772 Berlin, Germany
Telephone: + 49 30 2601-2652; Facsimile: + 49 30 2601-1723; E-mail: krystyna.passia@din.de;
HTTP://www.jtc1sc27.din.de/en

Common Criteria v4 CCDB

- TOE to leverage CAPEC & CWE
- Also investigating how to leverage ISO/IEC 15026's "Assurance Case" process

US (NIAP) Evaluation Scheme

- Above plus
- Also investigating how to leverage SCAP

New Work Item Proposal

NP submitting

PROPOSAL FOR A NEW WORK ITEM

Date of presentation of proposal: YYYY-MM-DD	Proposer: ISO/IEC JTC 1 SC27
Secretariat: National Body	**ISO/IEC JTC 1 N** XXXX ISO/IEC JTC 1/SC 27 N

A proposal for a new work item shall be submitted to the secretariat of the ISO/IEC joint technical committee concerned with a copy to the ISO Central Secretariat.

Presentation of the proposal

Title Secure software development and evaluation under ISO/IEC 15408 and ISO/IEC 18405

Scope

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

This Technical Report (TR) provides guidance for the developer and the evaluator on how to use the CAPEC as a technical reference point during the TOE development life cycle and in an evaluation of the TOE secure software under ISO/IEC 15408 and 18045, by addressing:

a) A refinement of the IS 15408 Attack Potential calculation table for software, taking into account the entries contained in the CAPEC and their characterization.
b) How the information for mitigating software common attack patterns and related weaknesses is used in an IS 15408 evaluation, in particular providing guidance on how to determine which attack patterns and weaknesses are applicable to the TOE, taking into consideration of
    1. the TOE technology;
    2. the TOE security problem definition;
    3. the interfaces the TOE exports that can be used by potential attackers;
    4. the Attack Potential that the TOE needs to provide resistance for.
c) How the technical rationale provided by the developer for mitigating software common attack patterns and related weaknesses is used in the evaluation of the TOE design and the development of test cases.
d) How the CAPEC and related Common Weakness Enumeration (CWE) taxonomies are used by the evaluator, who needs to consider all the applicable attack patterns and be able to exploit specific related software weaknesses while performing the subsequent vulnerability analysis (AVA_VAN) activities on the TOE.
e) How incomplete entries from the CAPEC are resolved during an IS 15408 evaluation.
f) How the evaluator's attack and weakness analysis of the TOE incorporates other attacks and weaknesses not yet documented in the CAPEC.

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

ISO/IEC JTC 1/SC 7/WG 3, NWP "Refining Software Vulnerability Analysis Under ISO/IEC 15408 and ISO/IEC 18045"

- The way how the CAPEC and related CWE taxonomies are to be used by the developer, which needs to consider and provide sufficient and effective mitigation to all applicable attacks and weaknesses.

- The way how the CAPEC and related CWE taxonomies are to be used by the evaluator, which needs to consider all the applicable attack patterns and be able to exploit all the related software weaknesses while performing the subsequent AVA_VAN activities.

- How incomplete entries from the CAPEC are to be addressed during an evaluation.

- How to incorporate to the evaluation attacks and weaknesses not included in the CAPEC.

Status of ITU-T Recommendations

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

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

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

Industry Uptake

Foreword

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

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

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

It is important to note that these are the "practiced practices" employed by SAFECode members, which we identified through an ongoing analysis of our members' individual software security efforts. By bringing these methods together and sharing them with the larger community, SAFECode hopes to move the industry beyond defining theoretical best practices to describing sets of software engineering practices that have been shown to improve the security of software and are currently in use at leading software companies. Using this approach enables SAFECode to encourag... best practices that are prove... to ... and implementable even when ... requirements and development ... taken into account.

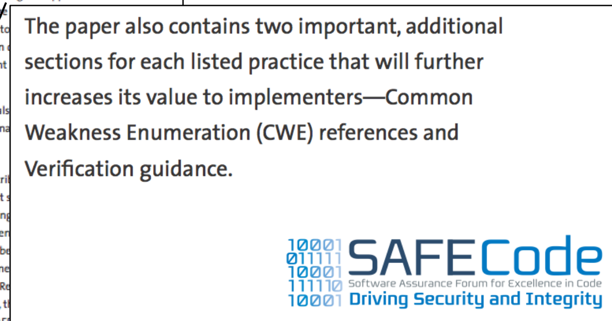
Though expanded, our key goals remain—keep it concise, actiona...

What's New

This edition of the paper prescri... updated security practices that ... during the Design, Programming ... ties of the software developmen... practices have been shown to be ... diverse development environme... original also covered Training, Re... Handling and Documentation, t... given detailed treatment in SAF... security engineering training and software integrity in the global supply chain, and thus we have refined our focus in this paper to concentrate on the core areas of design, development and testing.

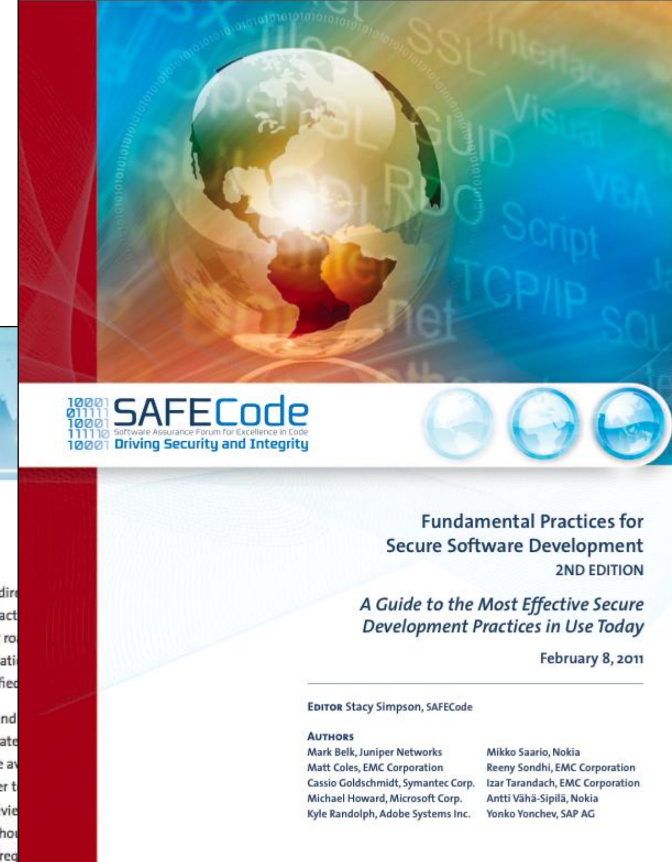
The paper also contains two important, additional sections for each listed practice that will further increases its value to implementers—Common Weakness Enumeration (CWE) references and Verification guidance.

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



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

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



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

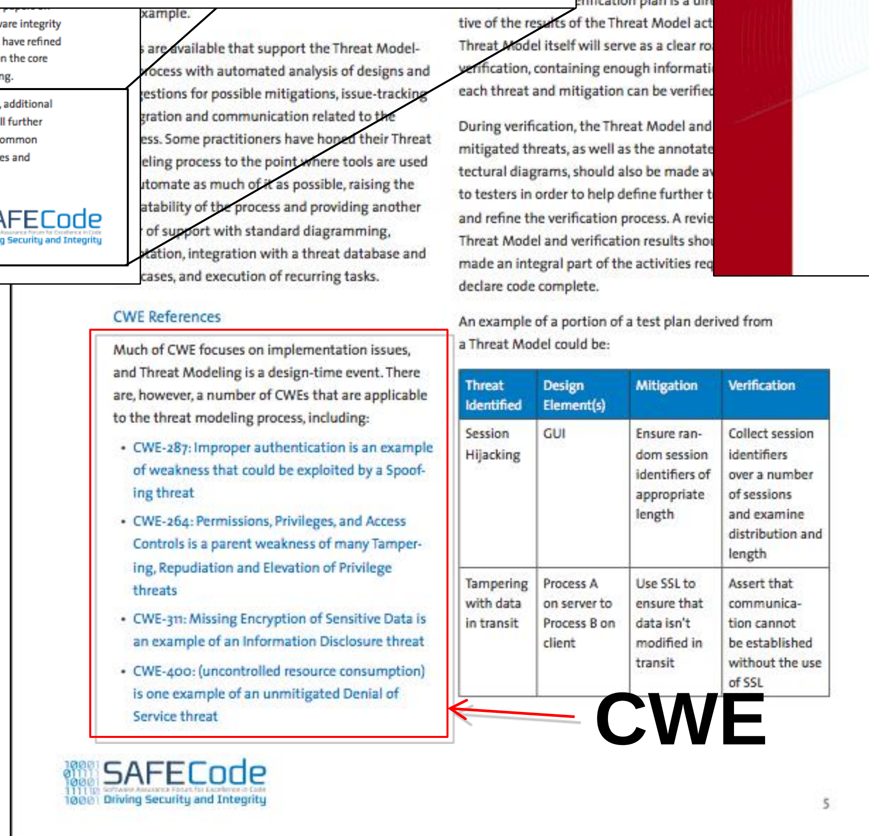
Fundamental Practices for Secure Software Development
2ND EDITION

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

February 8, 2011

Editor Stacy Simpson, SAFECode

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



...xample.

...s are available that support the Threat Model- ...rocess with automated analysis of designs and ...estions for possible mitigations, issue-tracking ...gration and communication related to the ...ess. Some practitioners have honed their Threat ...eling process to the point where tools are used ...tomate as much of it as possible, raising the ...atability of the process and providing another ... of support with standard diagramming, ...ation, integration with a threat database and ...cases, and execution of recurring tasks.

CWE References

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

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

...erification plan is a dir... tive of the results of the Threat Model act... Threat Model itself will serve as a clear ro... verification, containing enough informati... each threat and mitigation can be verified...

During verification, the Threat Model and ... mitigated threats, as well as the annotate... tectural diagrams, should also be made a... to testers in order to help define further t... and refine the verification process. A revie... Threat Model and verification results sho... made an integral part of the activities req... declare code complete.

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

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

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

CWE

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 vuln 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 a 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 infor
6 For more infor

IBM

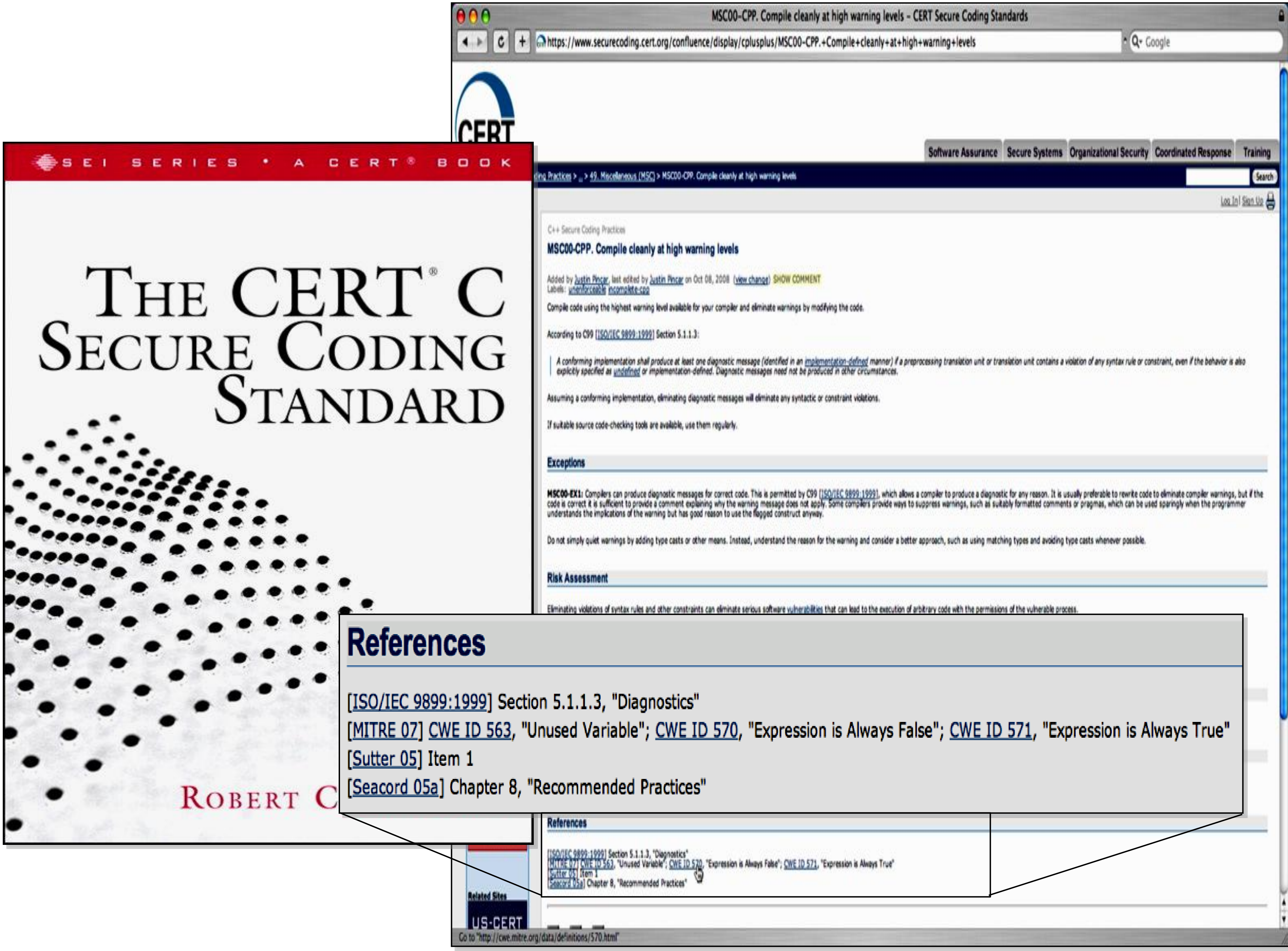
Security in Development: The IBM Secure Engineering Framework

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


SEI SERIES • A CERT® BOOK
THE CERT® C SECURE CODING STANDARD
ROBERT C
MSC00-CPP. Compile cleanly at high warning levels - CERT Secure Coding Standards
https://www.securecoding.cert.org/confluence/display/cplusplus/MSC00-CPP.+Compile+cleanly+at+high+warning+levels
CERT
Software Assurance
Secure Systems
Organizational Security
Coordinated Response
Training
C++ Secure Coding Practices
MSC00-CPP. Compile cleanly at high warning levels
Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.
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"
US-CERT

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 2006. It is based on the CWE (Common Weakness Enumeration) that provides a unified, measurable set of software weaknesses that will enable more effective discussion and action to find these weaknesses in source code and eliminate them. The CWE was developed by MITRE and sponsored by the Department of Homeland Security. The numbers between parentheses represent the common weakness enumeration IDs for each weakness. Numbers between square brackets are direct children of the weakness listed. CWE IDs can be found at the MITRE CWE Website or accessed directly by putting the number (in place of ###) in the following URL:

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

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 Signature
- Use of Less Trusted Source
- Acceptance of Extraneous Untrusted Data With Trusted Data
- Improperly Trusted Reverse DNS
- Insufficient Type Distinction
- Cross-Site Request Forgery (CSRF) - (352)
- Failure to Add Integrity Check Value
- Improper Validation of Integrity Check Value
- Trust of System Event Data
- Reliance on File Name or Extension of Externally-Supplied File
- Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking

Privacy Violation

Reliance on Cookies without Validation and Integrity Checking

Client-Side Enforcement of Server-Side Security - (602)

Improperly Implemented Security Check for Standard

Improper Authentication

User Interface Security Issues

Use of Insufficiently Random Values - (330)

Logging of Excessive Data

Certificate Issues

Cryptographic Issues

- Key Management Errors
- Missing Required Cryptographic Step
- Not Using a Random IV with CBC Mode
- Failure to Encrypt Sensitive Data
  - Cleartext Storage of Sensitive Information
  - Cleartext Transmission of Sensitive Information - (319)
  - Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
- Reversible One-Way Hash
- Inadequate Encryption Strength
  - Use of a Broken or Risky Cryptographic Algorithm - (327)
- Use of RSA Algorithm without OAEP

Permissions, Privileges, and Access Controls

- Access Control (Authorization) Issues - (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

The Tech Beat

NSA, DHS, Industry Gang Up on Dangerous Software Errors

Posted by: Stephen Wildstrom on January 12

Computer security experts have warned for years that the endless cycle of software flaws and exploits will only be broken when we create incentives for software authors and publishers to get it right. On Jan. 12, the industry took a potentially important step toward that goal when a broad coalition of companies, government agencies, academics, and advocacy groups launched a program to assure than software is free of 25 common errors that lead to the bulk of security problems.

The key to making the program effective is that it goes well beyond recommending best practices. Software buyers, particularly governments and large corporations are being urged to demand that vendors certify that code they sell is free of these 25 errors, and there's nothing like potential legal liability to get a company's attention. In addition, colleges are pledging to train students in writing software and employers can use the guidelines to assess the skills of

2010 CWE/SANS Top 25 Programming Errors List Receives Extensive News Coverage

CWE and the SANS Institute posted the completed 2010 CWE/SANS Top 25 Programming Errors on the CWE and SANS Web sites on February 16, 2010. A collaboration between the SANS Institute, MITRE, and over 40 top software security experts in the U.S. and Europe, the list provides detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigating and avoiding them.

The release received extensive news media coverage:

- *Top 25 Most Dangerous Programming Errors Revealed, InformationWeek, February 16, 2010*
- *Hold vendors liable for buggy software, group says, Computerworld, February 16, 2010*
- *Top 25 Programming Errors Highlight Application Security Challenges, eWeek, February 16, 2010*

CWE and Top 25 Are Main Topics of *Federal News Radio* Interview

Federal News Radio interviewed CWE/CAPEC Program Manager Robert A. Martin on March 10, 2010 about CWE and the Top 25 Most Dangerous Programming Errors. In the interview, entitled "Top federal software security holes exposed," Martin states: "The big problem is that traditional education in our country and across the world for software developers, for testers, for program managers has pretty much ignored this area. We put them into our software because we didn't know how they happened. So the CWE, the full Common Weakness Enumeration and then this prioritized part of the CWE, which we're calling the Top 25 Most Dangerous Programming Errors is basically an education tool at the first level. These are issues you should be aware of. You should ask your developers "have you been trained to recognize these if someone puts them in accidentally? Do you know how to program around these so that you don't introduce them?" You test people. "Do you know how to try to misuse and abuse your system?" So that, if there any of these latently in your software, you can find them before the user has it in his hands." A summary of the interview was published on the Federal News Radio Web site.

CWE Mentioned in *Federal News Radio* Interview about Software Assurance

Federal News Radio interviewed Joe Jarzombek, director for software assurance in the National Cyber Security Division of the Department of Homeland Security, on March 3, 2010 about software assurance. In the interview, entitled, "Software assurance affects more than just programmers," Jarzombek "explains why the CWE benefits cyber security -- and why this impacts more than programmers." A summary of the interview was published on the Federal News Radio Web site.

- *MITRE, List of 2010's Most Dangerous Programming Errors, TheServerSide.com, February 16, 2010*
- *Top 25 Programming Errors: Should Software Developers be Liable?, Bankinfo Security, February 16, 2010*
- *Is It Time For Software Liability?,Information Week, February 16, 2010*
- *Group Proposes Suits Over Faulty Code, Gov Info Security, February 16, 2010*
- *The top 25 most dangerous programming errors, Sunbelt Blog, February 16, 2010*
- *Experts: Developers Responsible for Programming Problems, Computerworld/IDG Norway, February 16, 2010*

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

SDL and the CWE/SANS Top 25

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

May.

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

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

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

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

CWE Outreach: A Team Sport

May/June Issue of IEEE Security & Privacy…

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

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 >>

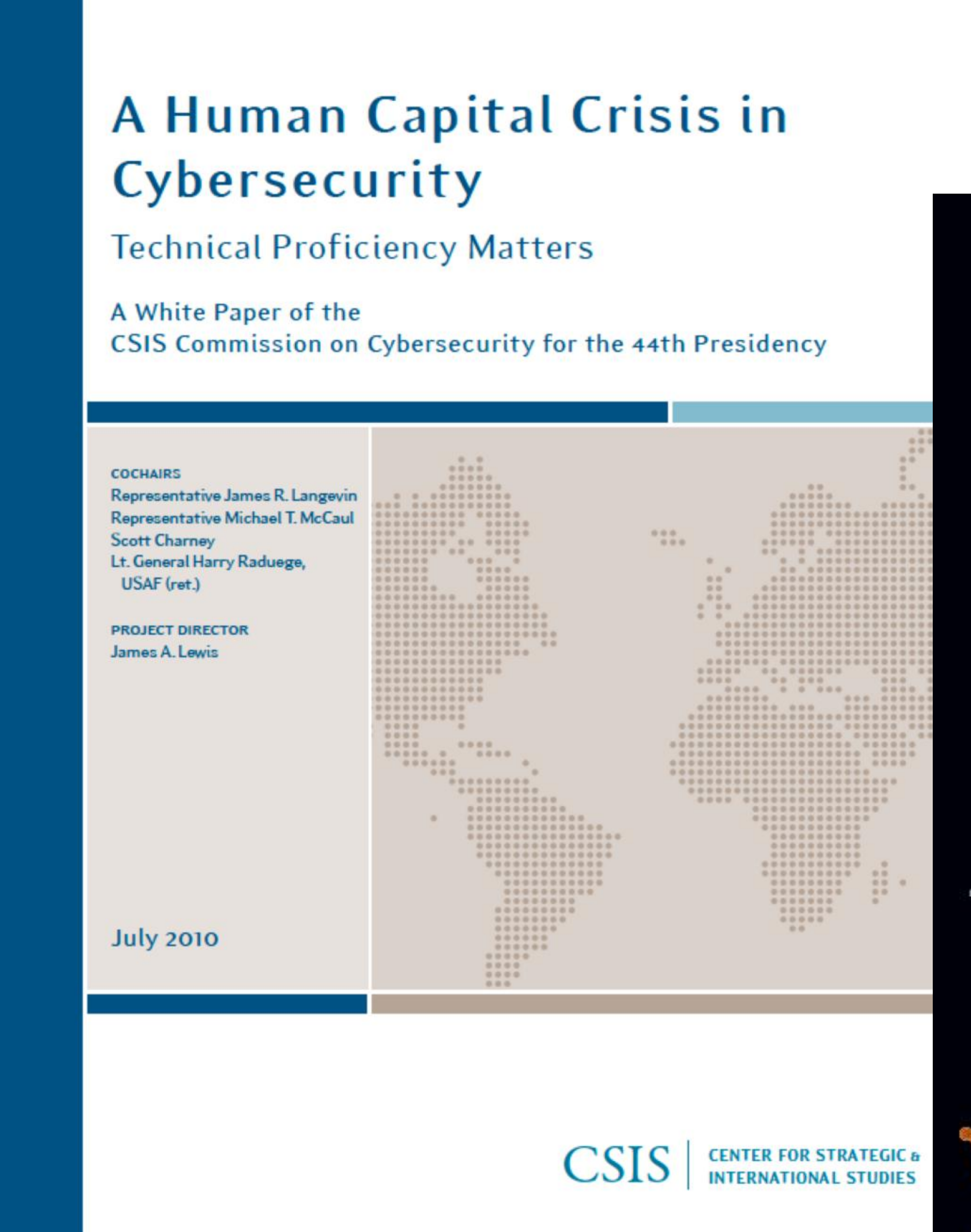
How do attackers exploit the lack of this control?

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 b…

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.


A Human Capital Crisis in Cybersecurity

Technical Proficiency Matters

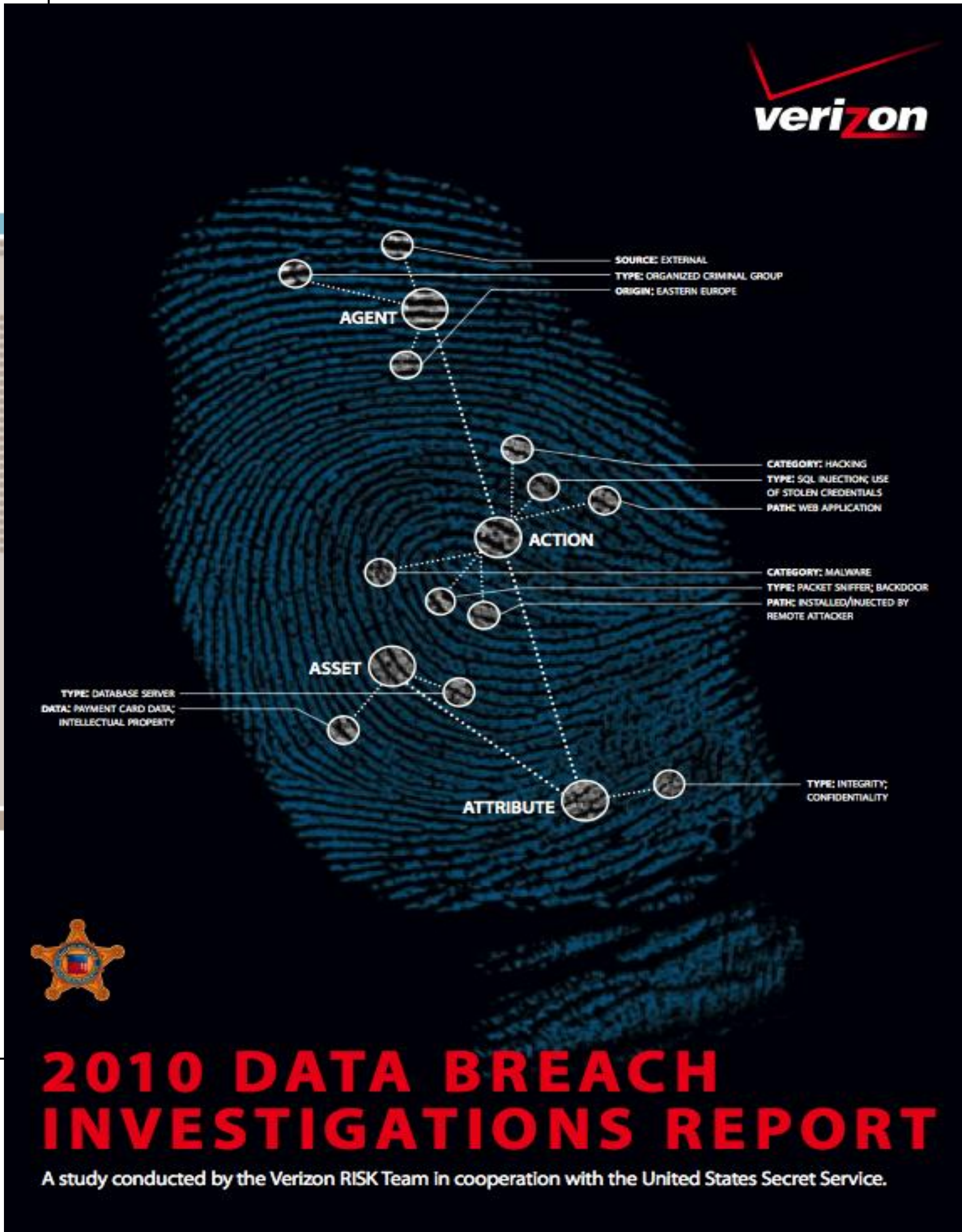
A White Paper of the
CSIS Commission on Cybersecurity for the 44th Presidency

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

PROJECT DIRECTOR
James A. Lewis

July 2010

CSIS | CENTER FOR STRATEGIC & INTERNATIONAL STUDIES



Questions?

ramartin@mitre.org