Being Explicit About Weaknesses

Robert A. Martin - MITRE
Sean Barnum - Cigital
Steve Christey - MITRE

1 March 2007
Software Security Assurance

Don't Let Your Applications Get You Down

Exploited software flaws cost the U.S. financial services industry more than $3 billion per year, according to the National Institute of Standards & Technology.

BY JOHN K. WATERS

Most enterprises have figured out that protecting software and hardware systems, although easy to compute in their complexity, is often overlooked in security posture. The result is a growing number of security and exploitable weaknesses in applications.

These vulnerabilities are costing enterprises a lot. A recent study by the National Institute of Standards & Technology shows that companies are spending $4 billion per year in software flaws, and over $1 billion per year in software errors. Exploited software is less than a quarter of the total costs in the financial services industry more than 10 years, according to NIST.

Part of the problem is the changing nature of enterprise computing. Applications, once kept in centralized data centers and accessible only by users in the enterprise's network, are now connected to customers, partners and suppliers through the Web.

Another problem is the nature of development. In today's application development, developers are rewarded for delivering features and meeting deadlines. This has led developers to focus on the business. Consequently, security becomes an afterthought, and vulnerabilities are discovered and addressed only if they are discovered before applications are deployed.

Not doing the security work up front can cost enterprises a lot, especially. Gartner figures it's an enormous cost.
Software Assurance

Background

- In October 2002, the President’s Critical Infrastructure Protection Board (PCIPB) created the National Security Agency (NSA) - led IT Security Study Group (ITSSG) to review existing IT acquisition processes.
- In July 2003, the Assistant Secretary of Defense for Networks and Information Integration (ASD(NII)) established the Software Assurance Initiative to examine software assurance issues.
- On 23 Dec 04, Undersecretary of Defense for Acquisitions, Technology and Logistics (USD(AT&L)) and ASD(NII) established a Software Assurance (SwA) Tiger Team to:
  - Develop a holistic strategy to reduce SwA risks within 90 days
  - Provide a comprehensive briefing of findings, strategy, and plan
- Tiger Team presented its strategy to USD(AT&L) and ASD(NII) on March 28, and on May 2 was tasked to proceed with day 180 implementation plan.

Requirements

- What functional statements in OSD Guidance for SwA requirements best enable optimal vendor solutions?
  - Require higher level written policy to specify need for SwA requirements
  - “Compelling arguments and evidence that...comparatively with risk”
- Written SwA Principles in policy
  - Looked at 8500, 5000.2, 6000, 5170, 6212, ...
  - In 8500.2 Annex language to potentially leverage for SwA:
    - “…use IA best practices…”
    - “…software will be well bebehaved…”
    - Point to language in contracts
  - Contract language to show equivalence to ISO 15026 practices
  - Burden on PMO to understand and have confidence in level of SwA
  - Requirement in policy that whenever a new risk is ID’d or an old risk changes, contractor must be notified

Basis for SwA Technology

- Offensiveside
  - Pedigree problem
  - Interconnectivity
  - Value of IT Information
  - Many untrusted
- Defensive side
  - Fragmented efforts
  - Immature science
  - Lack of resources

Hard, complex problems unlikely to be solved by government in near term.

Driving Needs for Software Assurance

- Software vulnerabilities jeopardize intellectual property, business operations and services, infrastructure operations, and casualty.
- Growing awareness and concern over the ability of an adversary to subvert the software supply chain.
  - Federal Government relies on COTS products and commercial devices foreign and non-verified domestic suppliers to meet majority of IT needs.
  - Software development offers opportunities to insert malicious code or exploit vulnerabilities.
- Growing concern about inadequacies of suppliers’ capabilities and deliver secure software with rigorous levels of integrity.
  - Current education & training provides too few practitioners with required competencies in secure software engineering.
- Concern about suppliers not exercising “minimum level of responsibility.
- Growing need to improve both the state-of-the-practice and the state-of-the-art in software capabilities of the nation.
- Processes and technologies are required to build trust into software acquired and used by Government and critical infrastructure.

Software Assurance (SwA) Definition

Software assurance (SwA) is the level of confidence that software is free of exploitable vulnerabilities, either intentionally or unintentionally designed as part of the software or inadvertently created.
Motivation for Classes of Software Security Flaws & Vulnerabilities

- For Systematic Study – classify security problems in software into categories that one can dissect for systematic study.
- For SS Tools Evaluation - a taxonomy of security vulnerability that the SA community would agree upon will be essential for evaluating Software Security (SS) tools and classifying SA functions.
- For SRD Development - Classes of software security flaws and vulnerabilities is one of resources to drive a standard reference dataset, which, in simply put, is a benchmark test suite for Software Security tools.

Possible Goals of Classifying Software Security Flaws & Vulnerabilities

- A taxonomy that has classification categories with the satisfactory characteristics as possible.
- Incorporate commonly used terms in security vulnerabilities that occurred in modern days.
- Consensus from the SA community.
Information Technology Laboratory
Software Diagnostics and Conformance Testing Division

Workshop on
Software Security Assurance Tools, Techniques, and Metrics

PROGRAM
November 7, 2005
8:30 – 9:00 : Welcome – Paul E. Black
9:00 – 10:30 : Tools and Metrics - Liz Fong
   • Where do Software Security Assurance Tools Add Value? – David Jackson, David Cooper
   • Metrics that Matter – Brian Chess
   • The Case for Common Flaw Enumeration – Robert Martin, Steven Christey, Joe Jarzombek
10:00 – 11:00 : Break
11:00 – 12:30 : Flaw Taxonomy and Benchmarks - Robert Martin
   • Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors – Katrina Tsipenyuk, Brian Chess, Gary McGraw
   • A Taxonomy of Buffer Overflows for Evaluating Static and Dynamic Software Testing Tools – Kendra Kratkiewicz, Richard Lippmann
   • ABM – A Prototype for Benchmarking Source Code Analyzers – Tim Newsham, Brian Chess
Goal of the Common Weakness Enumeration Initiative

- To improve the quality of software with respect to known security issues within source code
  - define a unified measurable set of weaknesses
  - enable more effective discussion, description, selection and use of software security tools and services that can find these weaknesses
Clarifying software weaknesses: Enabling communication (1 of 2)

- Systems Development Manager Issue Areas:
  - What are the software weaknesses I need to protect against
    - Architecture, design, code
  - Can I look through the issues by technologies, risks, severity
  - What have the pieces of my system been vetted for?
    - COTS packages, organic development, open source
  - Identify tools to vet code based on tool coverage
    - How effective are the tools?

- Assessment Tool Vendors Issue Areas:
  - Express what my tool does
  - Succinctly identify areas I should expand coverage
Clarifying software weaknesses:

Enabling communication (2 of 2)

- **COTS Product Vendor Issue Areas:**
  - What have I vetted my applications for?
  - What do my customers want me to vet for?

- **Researcher Issue Areas:**
  - Quickly understand what is known
  - Easily identify areas to contribute/refine/correct

- **Educator Issue Areas:**
  - Train students with the same concepts they’ll use in practice

- **Operations Manager Issue Areas:**
  - What issues have my applications been vetted for? (COTS/Organic/OS)
  - What types of issues are more critical for my technology?
  - What types of issues are more likely to be successfully exploited?
CWE Launched March 2006 with draft 1, now at draft 5

[cwe.mitre.org]
Building Consensus About A Common Enumeration

Common Weakness Enumeration (CWE)
- call & count the same
  ● enable metrics

Previously Published Vulnerability Taxonomy Work
CVE-based PLOVER Work

GMU
Stanford
Purdue
SPI Dynamics
Kestrel Technology

Gramma Tech's Checklist and Taxonomy

Dictionary

OWASP's Checklist and Taxonomy

Microsoft's Mike Howard's Work and Taxonomy

Core Security
JMU
MIT LL
Unisys
Cenzic
KDM Analytics

GMU
SEI
VERACODE
UC Berkeley
Parasoft
Security Institute
KDM Analytics

CGI
t's
Gary
McGraw's
Work and
Taxonomy

Fortify's
Brian
Chess's
Work and
Taxonomy

Klocwork's Checklist and Taxonomy

OWASP's Checklist and Taxonomy

Secure Software's John Viega's CLASP and Taxonomy

Dictionary

Common Weakness Enumeration (CWE)
- call & count the same
  ● enable metrics
Status (as of Feb 1, 2007)
• 21,990 unique CVE names
Vulnerability Type Trends:
A Look at the CVE List (2001 - 2006)

- 15% “other”
But…

• What about the 15% “Other” in 2006?
  – What is up-and-coming? What’s important but below the radar?
• Variants matter in evaluating software quality
  – Example: obvious XSS vs. non-standard browser behaviors that bypass filters
• Bug X might be “resultant from” or “primary to” Bug Y, yet both are thought of as vulnerabilities
  – E.g. integer overflows leading to buffer overflows
  – How can we tell if things are improving?
• Maybe some issues are symptoms of deeper problems
  – Error: Couldn’t open file “lang-
    <SCRIPT>alert(‘XSS’)</SCRIPT>.txt”
Removing and Preventing the Vulnerabilities Requires More Specific Definitions...

Cross-site scripting (XSS):
- Basic XSS
- XSS in error pages
- Script in IMG tags
- XSS using Script in Attributes
- XSS using Script Via Encoded URI Schemes
- Doubled character XSS manipulations, e.g. `&lt;script`
- Invalid Characters in Identifiers
- Alternate XSS syntax

Buffer Errors
- Unbounded Transfer ('classic overflow')
- Write-what-where condition
- Boundary beginning violation ('buffer underwrite')
- Out-of-bounds Read
- Wrap-around error
- Unchecked array indexing
- Length Parameter Inconsistency
- Other length calculation error
- Miscalculated null termination
- String Errors

Relative Path Traversal
- Path Issue - dot dot slash - `../../filedir`
- Path Issue - leading dot dot slash - `../filedir`
- Path Issue - leading directory dot dot slash - `directory/../../filedir`
- Path Issue - directory doubled dot dot slash - `directory/./filename`
- Path Issue - dot dot backslash - `\./filename`
- Path Issue - leading dot dot backslash - `\..\..\filename`
- Path Issue - directory doubled dot dot backslash - `directory/./..\filename`
- Path Issue - dot dot backslash - `\..\filename`
- Path Issue - leading directory dot dot backslash - `directory\./filename`
- Path Issue - directory doubled dot dot backslash - `directory\./..\filename`
- Path Issue - triple dot - `...`
- Path Issue - multiple dot - `....`
- Path Issue - doubled dot dot slash - `.../.../`
- Path Issue - doubled triple dot slash - `.../.../`
… which led to the Preliminary List of Vulnerability Examples for Researchers (PLOVER)

- Initial goal: extend vulnerability auditing checklist
- Collected extensive CVE examples
  - Emphasis on 2005 and 2006
  - Reviewed all issues flagged "other"
- 300 weakness types, 1500 real-world CVE examples
- Identified classification difficulties
  - Primary vs. resultant vulns
  - Multi-factor issues
  - Uncategorized examples
  - Tried to separate attacks from vulnerabilities
- Beginning vulnerability theory
  - Properties
  - Manipulations
  - Consequences
- One of the 3 major sources of CWE
**PLOVER:**
300 “types” of Weaknesses, 1500 real-world CVE examples

<table>
<thead>
<tr>
<th>Category</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BUFF] Buffer overflows, format strings, etc.</td>
<td>10</td>
</tr>
<tr>
<td>[SVM] Structure and Validity Problems</td>
<td>10</td>
</tr>
<tr>
<td>[SPEC] Special Elements (Characters or Reserved Words)</td>
<td>19</td>
</tr>
<tr>
<td>[SPECM] Common Special Element Manipulations</td>
<td>11</td>
</tr>
<tr>
<td>[SPECTS] Technology-Specific Special Elements</td>
<td>17</td>
</tr>
<tr>
<td>[PATH] Pathname Traversal and Equivalence Errors</td>
<td>47</td>
</tr>
<tr>
<td>[CP] Channel and Path Errors</td>
<td>13</td>
</tr>
<tr>
<td>[CCC] Cleansing, Canonicalization, and Comparison Errors</td>
<td>16</td>
</tr>
<tr>
<td>[INFO] Information Management Errors</td>
<td>19</td>
</tr>
<tr>
<td>[RACE] Race Conditions</td>
<td>6</td>
</tr>
<tr>
<td>[PPA] Permissions, Privileges, and ACLs</td>
<td>20</td>
</tr>
<tr>
<td>[HAND] Handler Errors</td>
<td>4</td>
</tr>
<tr>
<td>[UI] User Interface Errors</td>
<td>7</td>
</tr>
<tr>
<td>[INT] Interaction Errors</td>
<td>7</td>
</tr>
<tr>
<td>[INIT] Initialization and Cleanup Errors</td>
<td>6</td>
</tr>
<tr>
<td>[RES] Resource Management Errors</td>
<td>11</td>
</tr>
<tr>
<td>[NUM] Numeric Errors</td>
<td>6</td>
</tr>
<tr>
<td>[AUTHENT] Authentication Error</td>
<td>12</td>
</tr>
<tr>
<td>[CRYPTO] Cryptographic errors</td>
<td>13</td>
</tr>
<tr>
<td>[RAND] Randomness and Predictability</td>
<td>9</td>
</tr>
<tr>
<td>[CODE] Code Evaluation and Injection</td>
<td>4</td>
</tr>
<tr>
<td>[ERS] Error Conditions, Return Values, Status Codes</td>
<td>4</td>
</tr>
<tr>
<td>[VER] Insufficient Verification of Data</td>
<td>7</td>
</tr>
<tr>
<td>[MAID] Modification of Assumed-Immutable Data</td>
<td>2</td>
</tr>
<tr>
<td>[MAL] Product-Embedded Malicious Code</td>
<td>7</td>
</tr>
<tr>
<td>[ATTMIT] Common Attack Mitigation Failures</td>
<td>3</td>
</tr>
<tr>
<td>[CONT] Containment errors (container errors)</td>
<td>3</td>
</tr>
<tr>
<td>[MISC] Miscellaneous WIFFs</td>
<td>7</td>
</tr>
</tbody>
</table>
Vulnerability Theory:
Problem Statement and Rationale

- With 600+ variants, what are the main themes?
- Why is it so hard to classify vulnerabilities cleanly?
  - CWE, Pernicious Kingdoms, OWASP, others have had similar difficulties
- Same terminology used in multiple dimensions
  - Frequent mix of attacks, threats, weaknesses/faults, consequences
  - E.g. buffer overflows, directory traversal

- Goal: Increase understanding of vulnerabilities
  - Vocabulary for more precise discussion
  - Label current inconsistencies in terminology and taxonomy
  - Codify some of the researchers’ instinct

- One possible application: gap analysis, defense, and design recommendations
  - “Algorithms X and Y both assume input has property P. Attack pattern A manipulates P to compromise X. Would A succeed against Y?”
  - “Technology Z has properties P1 and P2. What vulnerability classes are most likely to be present?”
  - “Why is XSS so obvious but so hard to eradicate?”
Some Basic Concepts, By Example

Buffer overflow using long DNS response

Role: Attacker  Actor: User
1

Role: Attacker  Actor: Consultant
3

Role: Victim  Actor: Service
2

1) Attacker (as user) sends directive over Telnet channel: “Log me in”
2) Server (the target) sends directive over DNS channel: “Tell me IP’s hostname”
3) DNS consultant (controlled by attacker) returns hostname with property “>300 BYTES”
4) Buffer overflow activated
Artifact Labels

- **Interaction Point**
  - A relevant point within the product where a user interacts with the product

- **Intermediate Fault**
  - A behavior by the product that has not yet affected correctness, but will

- **Control Transfer Point**
  - The point where the program’s behavior changes from correct to incorrect

- **Activation Point**
  - The point where the “payload” is activated and performs the actions intended by the attacker

- **Resultant Fault**
  - A fault after a “Primary” fault that is also where incorrect behavior occurs
Artifact Labels - Example

```plaintext
print HTTPResponseHeader;
print "<title>Hello World</title>";
ftype = HTTP_Query_Param("type");
str = "/tmp";
strcat(str, ftype); strcat(str, ".dat");
handle = fileOpen(str, "read");
while((line=readFile(handle)))
{
    line=stripTags(line, "script");
    print line;
    print "<br>\n";
}
close(handle);
```
Building Consensus About A Common Enumeration

Previously Published Vulnerability Taxonomy Work

CVE-based PLOVER Work

Common Weakness Enumeration (CWE)
- call & count the same
  - enable metrics

Dictionary

OWASP's Checklist and Taxonomy

Cigital's Gary McGraw's Work and Taxonomy

Microsoft's Mike Howard's Work and Taxonomy

Fortify's Brian Chess's Work and Taxonomy

Klocwork's Checklist and Taxonomy

Ounce Lab's Checklist and Taxonomy

Gramma Tech's Checklist and Taxonomy

Secure Software's John Viega's CLASP and Taxonomy

Dictionary

IBM
SEI
NSA/CTC
Core Security
JMU
MIT LL
Unisys
Security Institute
Cenzic
KDM Analytics
GMU
Stanford
UC Berkeley
Purdue
SPI Dynamics
Kestrel Technology
Watchfire
Oracle
CVE-based
Parasoft

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary

Dictionary
Where Did We Start?

- Objective: To identify, integrate and effectively describe common software weaknesses known to the industry and software assurance community

- Leveraging taxonometric approach for list integration
  - Identify and review dozens of existing taxonomies
    - Academic and professional (Aslam, RISOS, Landwehr, Bishop, Protection Analysis, etc)
    - High level lists
      - OWASP Top 10, 19 Deadly Sins, WASC, etc.
    - In-depth practical
      - PLOVER, CLASP, 7 Pernicious Kingdoms
  - Create visualizations for effective comparison and analysis
  - Integrating taxonomies
    - Normalizing and deconfliction
    - Finding a proper balance between breadth & depth
Formalizing a Schema for Weaknesses

### Identifying Information
- CWE ID
- Name

### Describing Information
- Description
- Alternate Terms
- Demonstrative Examples
- Observed Examples
- Context Notes
- Source
- References

### Scoping & Delimiting Information
- Functional Area
- Likelihood of Exploit
- Common Consequences
- Enabling Factors for Exploitation
- Common Methods of Exploitation
- Applicable Platforms
- Time of Introduction

### Prescribing Information
- Potential Mitigations

### Enhancing Information
- Weakness Ordinality
- Causal Nature
- Related Weaknesses
- Taxonomy Mapping
- Research Gaps
CWE-79  Cross-site scripting (XSS)

CWE ID: 79

Description:
Cross-site scripting weakness occurs when dynamically generated web pages display input, such as login information, that is not properly validated, allowing an attacker to embed malicious scripts into the generated page and then execute the script on the machine of any user that views the site. If successful, Cross-site scripting vulnerabilities can allow an attacker to steal data or take control of the victim's system.

Alternate Terms:
"CSS" was once used, but it gives rise to confusion with the "CSS" language that formats web pages, and in practice, is used significantly, and its use is discouraged.

Likelihood of Exploit:
High to Very High

Weakness Ordinality:
Resultant (Weaknesses)

Causal Nature:
Explicit (This is an exploit fault).

Common Consequences:
Confidentiality: The misconfiguration of the site can lead to the exposure of sensitive user information.
Access control: In some cases, the attacker can steal or alter data on a victim's computer without knowledge of other flaws.

Potential Mitigations:
Carefully check each input field for injection, for example, by using a specification (white list) approach. All input should be sanitized, which may require you to specify, but all data input into the user interface or passed through headers, the URL itself, or the browser's history.

References:

Node Relationships:
Child Of: Injection (74)
Results In: Mobile Code: Invoking untrusted mobile code (494)
Parent Of: Basic XSS (80)
Parent Of: XSS in error pages (81)
Parent Of: Script in IMG tags (82)
Parent Of: XSS using Script in Attributes (83)
Parent Of: XSS using Script Via Encoded URI Schemes (84)
Parent Of: Doubled character XSS manipulations, e.g. '<<script' (85)
Parent Of: Invalid Characters in Identifiers (86)
Parent Of: Alternate XSS syntax (87)
Parent Of: Mobile Code: Invoking untrusted mobile code (494)

Source Taxonomies:
PLOVER - Cross-site scripting (XSS)
7 Pernicious Kingdoms - Cross-site Scripting
CLASP - Cross-site scripting

Applicable Platforms:
C
C++
Java
.NET
CWE Cross-Section:
20 of the Usual Suspects

- Absolute Path Traversal (CWE-36)
- Cross-site scripting (XSS) (CWE-79)
- Cross-Site Request Forgery (CSRF) (CWE-352)
- CRLF Injection (CWE-93)
- Error Message Information Leaks (CWE-209)
- Format string vulnerability (CWE-134)
- Hard-Coded Password (CWE-259)
- Insecure Default Permissions (CWE-276)
- Integer overflow (wrap or wraparound) (CWE-190)
- OS Command Injection (shell metacharacters) (CWE-78)
- PHP File Inclusion (CWE-98)
- Plaintext password Storage (CWE-256)
- Race condition (CWE-362)
- Relative Path Traversal (CWE-23)
- SQL injection (CWE-89)
- Unbounded Transfer ('classic buffer overflow') (CWE-120)
- UNIX symbolic link (symlink) following (CWE-61)
- Untrusted Search Path (CWE-426)
- Weak Encryption (CWE-326)
- Web Parameter Tampering (CWE-472)
CWE Cross-Section:
22 More Suspects

- Design-Related
  - High Algorithmic Complexity (CWE-407)
  - Origin Validation Error (CWE-346)
  - Small Space of Random Values (CWE-334)
  - Timing Discrepancy Information Leak (CWE-208)
  - Unprotected Windows Messaging Channel ('Shatter') (CWE-422)
  - Inherently Dangerous Functions, e.g. gets (CWE-242)
  - Logic/Time Bomb (CWE-511)

- Low-level coding
  - Assigning instead of comparing (CWE-481)
  - Double Free (CWE-415)
  - Null Dereference (CWE-476)
  - Unchecked array indexing (CWE-129)
  - Unchecked Return Value (CWE-252)
  - Path Equivalence - trailing dot - 'file.txt.' (CWE-42)

- Newer languages/frameworks
  - Deserialization of untrusted data (CWE-502)
  - Information leak through class cloning (CWE-498)
  - .NET Misconfiguration: Impersonation (CWE-520)
  - Passing mutable objects to an untrusted method (CWE-375)

- Security feature failures
  - Failure to check for certificate revocation (CWE-299)
  - Improperly Implemented Security Check for Standard (CWE-358)
  - Failure to check whether privileges were dropped successfully (CWE-273)
  - Incomplete Blacklist (CWE-184)
  - Use of hard-coded cryptographic key (CWE-321)

... and about 550 more
Where Are We Today?

Quality
- “Kitchen Sink” – In a good way
  - Many taxonomies, products, perspectives
  - Varying levels of abstraction
    - Directory traversal, XSS variants
- Mixes attack, behavior, feature, and flaw
  - Predominant in current research vocabulary, especially web application security
  - Complex behaviors don’t have simple terms
  - New/rare weaknesses don’t have terms

Quantity
- Draft 5 - over 600 entries
- Currently integrating content from top 15 – 20 tool vendors and security weaknesses “knowledge holders” under NDA

Accessibility
- Website is live with:
  - Historical materials, papers, alphabetical full enumeration, taxonomy HTML tree, CWE in XML, ability to URL reference individual CWEs, etc
Using A Unilateral NDA with MITRE to Bring in Info

Purpose:

- Sharing the proprietary/company confidential information contained in the underlying Knowledge Repository of the Knowledge Owner’s Capability for the sole purpose of establishing a public Common Weakness Enumeration (CWE) dictionary that can be used by vendors, customers, and researchers to describe software, design, and architecture related weaknesses that have security ramifications.

- The individual contributions from numerous organizations, based on their proprietary/company-confidential information, will be combined into a consolidated collection of weakness descriptions and definitions with the resultant collection being shared publicly.

- The consolidated collection of knowledge about weaknesses in software, design, and architecture will make no reference to the source of the information used to describe, define, and explain the individual weaknesses.
Coverage of CWE

Draft

55%

29%

9%

4%

1%

1%

1%

1%

none

one

two

three

four

five

six
Covered CWEs - By Number of Tools

- 67% one
- 20% two
- 9% three
- 2% four
- 1% five
- 1% six
Current Community Contributing to the Common Weakness Enumeration

- AppSIC
- Cenzic
- CERIAS/Purdue University
- CERT/CC
- Cigital
- CodescanLabs
- Core Security
- Coverity
- DHS
- Fortify
- IBM
- Interoperability Clearing House
- JHU/APL
- JMU
- Kestrel Technology
- KDM Analytics
- Klocwork
- McAfee/Foundstone
- Microsoft
- MIT Lincoln Labs
- MITRE
- North Carolina State University
- NIST
- NSA
- Oracle
- Ounce Labs
- OWASP
- Palamida
- Parasoft
- PolySpace Technologies
- proServices Corporation
- SecurityInnovation
- Secure Software
- Security University
- Semantic Designs
- SofCheck
- SPI Dynamics
- UNISYS
- VERACODE
- Watchfire
- WASC
- Whitehat Security, Inc.
- Tim Newsham

To join send e-mail to cwe@mitre.org
Planned Improvements - Content

- Metadata tagging
  - Language, OS, etc.
  - Time of Introduction
  - Vulnerability theory
  - Other ideas?

- Content cleanup
  - Consistent naming
  - Structural refactoring
  - Attack-centric wording (align to CAPEC)

- Formalization
  - SBVR
Planned Improvements - Site Usability

- **Search**
  - Select a subset of the catalog using any of the metadata
  - Display results and make available as XML
  - Predefined searches

- **Graphical Visualization**
  - Dynamic adjustment and navigation
  - Alternate taxonomies
Building Consensus About A Common Enumeration

Common Weakness Enumeration (CWE)
- call & count the same
- enable metrics

Previously Published Vulnerability Taxonomy Work
CVE-based PLOVER Work
Klocwork's Checklist and Taxonomy
Ounce Lab's Checklist and Taxonomy
OWASP's Checklist and Taxonomy
Cigital's Gary McGraw's Work and Taxonomy
Fortify's Brian Chess's Work and Taxonomy
Microsoft's Mike Howard's Work and Taxonomy
Gramma Tech's Checklist and Taxonomy
Core Security
JMU
Unisys
Security Institute
Cenzi
KDM Analytics
IBM
SEI
NSA/CTC
Core Security
JMU
Veracode
UC Berkeley
Parasoft
Watchfire
GMU
Stanford
Purdue
SPI Dynamics
Kestrel Technology
Oracle
Unisys
Security Institute
Cenzi
KDM Analytics
IBM
SEI
NSA/CTC
Core Security
JMU
Veracode
UC Berkeley
Parasoft
Watchfire
GMU
Stanford
Purdue
SPI Dynamics
Kestrel Technology
Oracle
CWE Compatibility and Effectiveness Program Launched
CWE Compatibility and Effectiveness Process Posted

CWE Compatibility and Effectiveness Program

Introduction
The CWE Compatibility and Effectiveness Program is a formal review and evaluation process for organizations wishing to declare their information security products and services as CWE-Compatible and CWE-Effective and have them formally evaluated.

Compatible and Effective products and services, as well as those working towards compatibility and effectiveness, will be posted on the "CWE-Compatible and Effective Products and Services" page on the CWE Web site and included on handouts at Information security and related trade shows and events at which MITRE exhibits CWE (see the CWE Calendar).

The formal CWE Compatibility and Effectiveness Program includes three phases: Declaration, Evaluation, and Effectiveness.

Phase 1 – Declaration Phase
The Declaration Phase requires the completion of a short informational "CWE Compatibility Declaration Form" used to register an organization’s declaration of intent with respect to CWE compatibility and effectiveness. In this phase you are asked to review the compatibility and effectiveness requirements and then make a statement regarding whether your organization believes that its product or service currently fulfills the compatibility requirements, or if your organization is working towards fulfilling the requirements. This phase of the CWE compatibility and effectiveness process does not result in an official evaluation or assessment by MITRE; rather, MITRE only reviews the declaration. As long as the products or services are commercially or publicly available, the declaration and an endorsement quote from you (if desired) is posted on the CWE Web site.

Phase 2 – Evaluation Phase
The Evaluation Phase requires completion of Phase 1 with "yes" as the answer for support of CWE output, CWE searchable, and CWE documentation. You must also complete an extended "CWE Compatibility Requirements Evaluation Form" that is a more extensive CWE-compatible formal review and includes several evaluation activities. You will also receive the "Compatible Product/Service Organization Welcome Kit" with items for your Web site.

This formal evaluation process includes a "branding program" and logo to indicate successful completion of the compatibility portion of the compatibility and effectiveness evaluation. A major component of this phase requires specific details about how your organization has satisfied each of the mandatory requirements in the Requirements and Recommendations for CWE Compatibility and CWE Effectiveness document. The Phase 2 "CWE Compatibility Requirements Evaluation Form" also requires the signature...
CWE Compatibility and Effectiveness Requirements Posted
CWE-Compatible & CWE-Effective

CWE Compatible:
1. CWE-compatible “intent” declared
   - vendor with shipping product declares intent to add support for CWE ids
2. CWE-compatible “output and searchable” declared
   - vendor declares that their shipping product provides CWE ids and supports searching
3. CWE-compatible “mapping accuracy” compatibility questionnaire posted
   - questionnaire for mapping accuracy posted to CWE web site
4. CWE-compatible means it meets the following requirements:
   - Can find items by CWE id (CWE searchable)
   - Includes CWE id in output for each item (CWE output)
   - Explain the CWE functionality in their item’s documentation (CWE documentation)
   - Provided MITRE with “weakness” item mappings to validate the accuracy of the product or services CWE ids
   - Makes a good faith effort to keep mappings accurate

CWE-Effective:
1. CWE-effectiveness list posted
   - CWE ids that the tool is declaring “effectiveness for” is posted to CWE web site
2. CWE-effectiveness test results posted
   - CWE test cases obtained from NIST reference data set generator by tool owner
   - Scoring sheet for requested CWE test cases provided to MITRE by NIST
   - Tool results from evaluating CWE-based sample applications (CWE test cases) provided to MITRE for processing and posting
The Road Ahead for the CWE effort

- Finish the strawman dictionary/taxonomy
- Create a web presence
- Get NDAs with knowledgeable organizations
- Merge information from NDA’d sources
- Get agreement on the detailed enumeration
- Dovetail with test cases (NIST/CAS)
- Dovetail with attack patterns (Cigital)
- Dovetail with coding standards (SEI CERT/CC)
- Dovetail with BSI, CBK, OMG SwA SIG, ISO/IEC,...
- Create alternate views into the CWE dictionary
- Establish CWE Editorial Board (roles & members)
- Establish CWE Compatibility Requirements
- Collect CWE Compatible Declarations