The Common Weakness Enumeration (CWE) Initiative

Part of the DHS/DoD Software Assurance Initiative’s Tools and Technologies Effort

CWE

cwe.mitre.org
[currently cve.mitre.org/cwe/]

June 27, 2006
Robert A. Martin
This Briefing Will Touch Upon Multiple Efforts Ongoing in the Software Assurance (SwA) Arena

- National Institute of Science and Technology (NIST)’s Software Assurance Metrics and Tool Evaluation (SAMATE)
- MITRE/Department of Homeland Security (DHS) Common Weakness Enumeration (CWE)
- Cigital/MITRE/DHS Common Attack Patterns Enumeration and Classification (CAPEC)
- Object Management Group (OMG) SwA Special Interest Group (SIG)
CVE Growth

Status
(as of May 17, 2006)
• 16,943 unique CVE names
Top Ten Vulnerability Types in CVE

(covers 2361 of 3933 CVE issues publicized between 1 Jan 2000-13 Feb 2003 inclusive)

- Buffer Overflow: 87
- Directory Traversal - dot dot: 26
- DOS by malformed input: 24
- Metachar Shell cmd: 17
- Cross-site Scripting: 14
- Unprotected Privileged Op's: 14
- Symbolic Link Following: 14
- Information Leak: 14
- Cryptographic error: 11
- Format String: 11

* The "Types" of Other and Unknown represent 831 vulnerabilities
## Preliminary List of Vulnerability Examples for Researchers (PLOVER)

300 “types” of Weaknesses

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

- Need for Flaw Taxonomy Identified as Supporting activity to NIST SAMATE effort to measure effectiveness of tools in finding these weaknesses
Defining a Software Security Flaw Taxonomy

Because of the nature of software development (new technologies, new languages and language features) a taxonomy of software security flaws will be a living and changing entity. Additional characteristics that must be considered include:

- Program vulnerabilities are usually a combination of security flaws

- Mutual flaw exclusion will be difficult to deal with (examples: authentication vs. logic flaw problem)

- Some of the flaws in the taxonomy cannot be identified by tools today

- Some flaws have never been seen in real world code… yet

- Some flaws can be introduced at multiple points in the SDLC
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
10:30 – 11:00: Break
11:00 – 12:30: Flaw Taxonomy and Benchmarks - Robert Martin

- Metrics that Matter – Brian Chess
- The Case for Common Flaw Enumeration – Robert Martin, Steven Christey, Joe Jarzombek
- 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 for the Common Weakness Enumeration:

- 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
What does CWE need to come to agreement on?

- Two separate but synergistic sub-goals:
  - Need detailed and specific definitions of the individual issues that we want to remove/reduce in software (a dictionary)
  - Need a structure/organization for thinking about the issues and allowing discussion/debate about entire groupings of issues (views/taxonomies)
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?
Building Consensus About A Common Enumeration

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

CWE Compatibility

DHS/NIST SAMATE Tool Assessment

List of CWEs that a Tool finds

DHS/NIST SAMATE Reference Dataset

NSA/CTC CAMP Reference Dataset

Secure Software’s John Viega’s CLASP 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

Klocwork’s Checklist and Taxonomy

Ounce Lab’s Taxonomy

CVE-based PLOVER Work

Previously Published Vulnerability Taxonomy Work

SwA Standard (MDA/ADM)

CVE and NVD using CWEs

Dictionary

CVE/cve.mitre.org

CWE
Common Weakness Enumeration (CWE)

DHS’s BSI Web site

NSA/CTC

Mitre

CVE

OMG
SwA Metrics & Tool Evaluation (SAMATE)

* SAMATE Reference Dataset (SRD), version 2, on-line
  This dataset will have 1000s of test cases for evaluation and development of SwA tools. Cases will have breadth of
  - language (C, Java, UML, etc.)
  - life cycle (design model, source code, application, ...)
  - size and type (small and huge, production and artificial, ...)
* Specifications and a reviewed test, including a suite of test cases (from the SRD above) for one class of SwA tool, probably source code scanners.
* Specifications & test for another class of SwA tool, probably web applications.
* Establish an advisory committee and create a road map to creating tests for all SwA tools (which tool classes should be done first?).
* List SwA areas with underdeveloped tools; sketch R&D that could fill each area.
* Requires Common Enumeration of Weaknesses (CWE) to provide a dictionary of software flaws

SAMATE project leader, Paul E. Black, paul.black@nist.gov (p.black@acm.org),
100 Bureau Drive, Stop 8970, Gaithersburg, Maryland  20899-8970
Current Community Contributing to the Common Flaw Enumeration

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

To join send e-mail to cwe@mitre.org
CWE 2nd Draft is available @ [cve.mitre.org/cwe]
Approximately 500 Dictionary Elements
Approximately 500 Dictionary Elements
The Classification Tree Nodes Link to the Dictionary Entries

CWE Classification Tree (initial draft)
- Common Weakness Enumeration and Classification
  - Location
  - Environment
  - Configuration
  - Code
    - Source Code
    - Data Handling
      - Input Validation
      - Output Validation
    - Range Errors
      - Buffer Errors
        - OVER - Unbounded Transfer ('classic overflow')
          - Stack overflow
          - Heap overflow
          - Write what-where conditions
        - UNDER - Boundary beginning violation ('buffer underwrite')
        - READ - Out-of-bound read
        - Wrap-around error
        - Unchecked array indexing
        - LEN - Length Parameter Inconsistency
        - LENCALC - Other length calculation error
        - Miscalculated null termination
        - String Errors
    - Type Errors
    - Representation Errors

CWE Dictionary (initial draft)

<table>
<thead>
<tr>
<th>Description</th>
<th>A stack overflow condition is a buffer overflow condition, where the buffer being overwritten is allocated on the stack (i.e., a local variable or, rarely, a parameter to a function).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likelihood of Exploit</td>
<td>Very high</td>
</tr>
<tr>
<td>Weakness Ordinarity</td>
<td>Primary (Weakness exists independent of other Weaknesses)</td>
</tr>
<tr>
<td>Causal Nature</td>
<td>Explicit (This is an explicit weakness resulting from behavior of the developer)</td>
</tr>
<tr>
<td>Common Consequences</td>
<td>Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop. Access control (memory and instruction processing): Buffer overflows can be used to execute arbitrary code, which is usually outside the scope of a program’s implicit security policy. Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.</td>
</tr>
<tr>
<td>Potential Mitigations</td>
<td>Pre-design: Use a language or compiler that performs automatic bounds checking. Design: Use an abstraction library to abstract away risky APIs. Not a complete solution. Pre-design through build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. Operational: Use OS-level preventative functionality. Not a complete solution.</td>
</tr>
</tbody>
</table>
What would be the details of the definitions?

- Name for an issue type
- Description of the type
- Description of the behavior of the issue
- Description of the exploit of the issue
- Description of the impact of the exploit
- Code samples for the languages/architectures where the issue exists
- CVE names of vulnerabilities of that issue type
- ...?

Currently CWE has:

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.
Common Attack Patterns Enumeration and Classification (CAPEC)

- **Description**
  - Supports classification taxonomies to be easily understood and consumable by the broad software assurance community and to be aligned and integrated with the other SwA community knowledge catalogs.

- **Tasks**
  - Identify and analyze reference Attack Pattern resources from academia, govnt, and industry.
  - Define standard Attack Pattern schema.
  - Identify and collect potential Attack Pattern seedling instances.
  - Finalize scope of effort to clarify number of Attack Patterns to be targeted for initial release.
  - Translate Attack Pattern seedling content into the defined schema.
  - Analyze and extend Attack Pattern seedlings to fulfill schema.
  - Identify set of new Attack Patterns to be authored.
  - Author targeted list of new Attack Patterns.
  - Map all Attack Patterns to the Common Weaknesses Enumeration (CWE).
  - Define a classification taxonomy for Attack Patterns.
  - Map Attack Patterns into the defined classification taxonomy.
  - Publish content to SwA community, solicit input, collaborate, review, and revise as needed.
  - Define process for ongoing extension and sustainment of the CAPEC.
  - Provide assistance to design, build, test, and deploy a website for public hosting of CAPEC.
The Challenge for the OMG SwA SIG:
How Do You Measure an Abstract Concept Like Secureness?
Standardization will ensure that all participants are investing not just in individual activities but in a coordinated strategy.

- Software Assurance (SwA) standardization will establish interoperability for exchange of information among participants
- Standardization of SwA means …
  - A formalization of the set of common definitions related to SwA
  - A format for exchanging information related to SwA
Assurance: Claims, Arguments, Evidence, Consequences, Risks

Evidence collected during development process: Automatically, semi-automatically, manually

SBVR

ISO 15026, Common Criteria, SOX, HIPAA, CWE, CVE

security requirements extraction from industry standards

SBVR, UML, QVT, XMI, KDM - ASTM

Risk evaluated based on Evidence within given context

SBVR

Metrics, Analysis

product evaluations

New product development

Input for new/improved software

Developers

Bug trucking System, Configuration Management, existing software systems

MDA

auto semi-auto manual

A D M

Consequences, Risks

Evidence collected during development process: Automatically, semi-automatically, manually

Assurance: Claims, Arguments, Evidence, Consequences, Risks

Assurance Leveraging Existing Standards

Metrics, Analysis

product evaluations

New product development

Input for new/improved software

Developers

Bug trucking System, Configuration Management, existing software systems

MDA

auto semi-auto manual

A D M

Evidence collected during development process: Automatically, semi-automatically, manually

Assurance: Claims, Arguments, Evidence, Consequences, Risks

Risk evaluated based on Evidence within given context

Leveraging Existing Standards

ISO 15026, Common Criteria, SOX, HIPAA, CWE, CVE

security requirements extraction from industry standards

SBO, UML, QVT, XMI, KDM - ASTM

Risk evaluated based on Evidence within given context

Assurance: Claims, Arguments, Evidence, Consequences, Risks

Evidence collected during development process: Automatically, semi-automatically, manually

Input for new/improved software

Developers

Bug trucking System, Configuration Management, existing software systems

M. D. A.

auto semi-auto manual

A. D. M.

Evidence collected during development process: Automatically, semi-automatically, manually

Assurance: Claims, Arguments, Evidence, Consequences, Risks

Risk evaluated based on Evidence within given context

Leveraging Existing Standards
The Road Ahead for the CWE effort

- Finishing the strawman dictionary/taxonomy
- Creating a web presence
- Getting NDAs with knowledgeable organizations
- Getting agreement on the detailed enumeration
- Dovetailing with test cases (NI ST/CAMP)
- Dovetailing with attack patterns (Cigital)
- Dovetailing with coding standards (SEI CERT/CC)
- Dovetailing with BSI, CBK, OMG SwA SIG, SC22,…
- Create alternate views into the CWE dictionary
## Acronyms from this Presentation

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM</td>
<td>Architecture Driven Modernization</td>
</tr>
<tr>
<td>BSI</td>
<td>Build Security In</td>
</tr>
<tr>
<td>CBK</td>
<td>Common Body of Knowledge</td>
</tr>
<tr>
<td>CVE</td>
<td>Common Vulnerabilities and Exposures</td>
</tr>
<tr>
<td>CWE</td>
<td>Common Weakness Enumeration</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off The Shelf</td>
</tr>
<tr>
<td>DHS</td>
<td>Department of Homeland Security</td>
</tr>
<tr>
<td>DITSCAP</td>
<td>DoD Information Technology Security Certification &amp; Accreditation Process</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>FISMA</td>
<td>Federal Information Security Management Act</td>
</tr>
<tr>
<td>HIPPA</td>
<td>Health Insurance Portability and Accountability Act</td>
</tr>
<tr>
<td>KDM</td>
<td>Knowledge Discovery Meta-Model</td>
</tr>
<tr>
<td>MDA</td>
<td>Model Driven Architecture</td>
</tr>
<tr>
<td>NDA</td>
<td>Non Disclosure Agreement</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Science and Technology</td>
</tr>
<tr>
<td>NSA</td>
<td>National Security Agency</td>
</tr>
<tr>
<td>NVD</td>
<td>National Vulnerability Database</td>
</tr>
<tr>
<td>OMG</td>
<td>Open Management Group</td>
</tr>
<tr>
<td>OWASP</td>
<td>Open Web Application Security Project</td>
</tr>
<tr>
<td>PLOVER</td>
<td>Preliminary List of Vulnerability Examples for Researchers</td>
</tr>
<tr>
<td>SAMATE</td>
<td>Software Assurance Measurement and Tool Evaluation</td>
</tr>
<tr>
<td>SIG</td>
<td>Special Interest Group</td>
</tr>
<tr>
<td>SOX</td>
<td>Sarbanes-Oxley</td>
</tr>
<tr>
<td>SPEM</td>
<td>Software Process Engineering Metamodel</td>
</tr>
<tr>
<td>SSATTM</td>
<td>Software Security Assurance Tools, Techniques, and Metrics</td>
</tr>
<tr>
<td>SwA</td>
<td>Software Assurance</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>