Goal of the Common Weakness Enumeration Initiative

- To improve the quality of software with respect to known security issues within code, design, or architecture
  - 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
Difficult to Understand & Act on Application Security Issues

CWE provides a mechanism for tying together information about application security
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.
Timeline of Items Enumerated and Defined in CWE

Incorporate weakness definitions from contributing organizations

# of Identified Items

# of “fully” Defined Items

CWE material from multiple sources - variety of aspects covered with some difference in terminology

Come to agreement on what the different aspects of a weakness need to be captured in the definition -- re-baseline “fully”

© 2007 MITRE
Formalizing a Schema for Weaknesses

**Identifying Information**
- CWE ID
- Name

**Describing Information**
- Description
- Extended Description
- Alternate Terms
- Demonstrative Examples
- Observed Examples
- Context Notes
- Source Taxonomy
- References
- Whitebox Definition
- Blackbox Definition
- Formal Definition

**Scoping & Delimiting Information**
- Type
- 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
- Affected Resource
- Related Attacks
- Obscurity Vector
- Node Relationships
- Research Gaps

---

**Scrubbing CWE**

- What do we have?
- If it’s not a weakness:
  - What is it?
  - What purpose is served by including it?
    - Who are the stakeholders
  - What’s the best way to handle it?
- If it’s related to other issues:
  - Why?
  - Are there multiple types of relationships?
  - Are there different perspectives?
- Is it too high level? To low level?
  - And to whom?
- If it just “feels” wrong
  - Why?
  - What should we do about it?
- See “Discussion of Issues in CWE Draft 6” for other examples

© 2007 MITRE
CWE “Scrub” Approach

- Review all 600+ nodes
  - At least 2 team members
  - Ask questions previously mentioned
  - Identify minor, major, and systemic issues
    - Most nodes had major or systemic issues
- Document high-level problems (“Discussion Points”)
  - Forces us to invent terminology
- Identify types of stakeholders, tasks, views

Approach (cont’d)

- Engage the community in discussions for systemic issues
- Modify schema
- Create several views
- Perform minor/major edits
- Prioritize nodes:
  - Feedback from external parties e.g. NSA
  - IDs being used by SAMATE, OWASP Top Ten, and NVD
CWE-79  Cross-site scripting (XSS)

[cwe.mitre.org/data/definition/79.html]
CWE Draft 8 - initial scope

- Add difference reports for each new release of CWE
- Rewrite all “attack-ish” weaknesses to be about the weakness (i.e., xss, csrf)
- Add weakness covered by CAPEC but not yet in CWE
- Add direct references to CAPEC IDs for attacks relevant to CWEs
- Other scrub-based clean ups as available

CWE Compatibility & Effectiveness Program (launched Feb 2007)

cwe.mitre.org/compatible/
Public Difference Reports Being Added
Behaviors, Properties, and Manipulations

A PRODUCT implements FEATURES by performing certain BEHAVIORS that operate on RESOURCES. The behaviors can be
SIMPLIFIED or COMPLEX, depending on the level of detail used at the time of implementation.

The product’s architecture may use caching, cookies, headers, referred-sites, and other features to handle resources. If so, these features may be used to implement behaviors that are relevant to the product.

The developer must ensure that any PUBLIC behavior is appropriate for the product’s intended policy. Policies are the intended behaviors that the product is expected to handle.

Types of Manipulations

There are three main manipulation types:

REACHABILITY - required to reach the relevant behavior. Example - when a buffer overflow can only occur in the password field, the reachability condition would be that the input must be longer than 10 characters.

TRIGGER - required to trigger the relevant behavior. Example - when a buffer overflow can only occur in the password field, the trigger condition would be that the input must be longer than 10 characters.

CROSSOVER - required to cross over into the relevant behavior. Example - when a buffer overflow can only occur in the password field, the crossover condition would be that the input must be longer than 10 characters.

Artifact Labels

Artifact Labels are used to mark important locations in code, design, or algorithm that are relevant to a vulnerability. Vulnerability researchers frequently highlight these locations when presenting vulnerable code, but they don’t use the same terminology, if any. These labels turn out to be useful in describing certain vulnerability “topology” in the abstract sense, as well.

NOTE: recommendations for alternate terms are welcome.

INTERACTION POINT - a location in the product where “input” (of either data or directives) enters the system. “Injection” might be a more natural term, but it’s already overloaded, and it seems to be data-centric. This is equivalent to what others call “entry points,” but that term has different uses in binary reverse engineering.

CROSSOVER POINT - the location in which an expected property is violated. Any subsequent behaviors that depend on the expected property could be subject to a vulnerable condition. A crossover point could occur between “lines” of code, e.g., if a product-generated filename is not checked for directory traversal sequences.

TRIGGER POINT - the location in the product where a “fault” occurs, and the product can no longer stop itself from performing incorrect behaviors in the future.

ACTIVATION POINT - the location where the attacker’s “payload” becomes activated; presumably the payload involves the incorrect behavior.

ATTACK VECTOR - a type of (PRIMARY FAULT, RESULTANT Faults, INTERACTION POINT, CROSSOVER POINT, TRIGGER POINT, ACTIVATION POINT). Different attacks could have different trigger and activation points; for example, a buffer overflow intended for DoS would have a different activation point than one intended for code execution.

Code Example

```c
1   print HTTPResponseHeader;
2   print '<title>World</title>';
3   strtype = HTTP_Query_Param('type');
4   str = '/tmp';
5   strcat(str, strtype); strcat(str, '.dat');
6   handle = fopen(str, 'r');
7   while((line = fgets(handle)))
8     {
9       line = stripTags(line, 'script');
10      print line;
11      print '<src\\n';
12     }
13   close(handle);
```

XSS:

- Interaction: 3, 6
- Intermediate Fault: 9
- Crossover: between 9 and 10
- Trigger: 10
- Activation: outside of program (when victim views page)

Traversal:

- Interaction: 3
- Crossover: between 5 and 6
- Trigger: 6
- Activation: 7 (or 10, depending on attack)

Overflow:

- Interaction: 3
- Crossover: between 4 and 5
- Trigger: 5
- Activation: 5 (if DoS intended), outside code (if code execution)
Current Community Contributing to the Common Weakness Enumeration

- AppSIC
- Apple
- Aspect Security
- Booz Allen Hamilton Inc.
- Cenzic
- CERIAS/Purdue University
- CERT/CC
- Codasafe
- Core Security
- Covertly
- Fortify
- Gramma Tech
- IBM
- Interoperability Clearing House
- JHU/APL
- JMU
- Kestrel Technology
- KDM Analytics
- Klocwork
- McAfee
- Microsoft
- MIT Lincoln Labs
- MITRE
- North Carolina State University
- NIST
- NSA
- OMG
- Oracle
- Ounce Labs
- OWASP
- Palamida
- Parasoft
- PolySpace Technologies
- proServices Corporation
- SANS Institute
- SecurityInnovation
- Secure Software
- Security University
- Semantic Designs
- SoCheck
- SPI Dynamics
- SureLogic, Inc.
- Symantec
- UNISYS
- VERACODE
- Watchfire
- WASC
- Whitehat Security, Inc.
- Tim Newsham

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

© 2007 MITRE

CWE “Scrub”
CWE Stakeholders

- Assessment Vendors
- Assessment Customers
- Software Developers
- Software Customers
- Academic Researchers
- Applied Vulnerability Researchers
- Refined Vulnerability Information Providers
- Educators
- Specialized Communities:
  - Web application security community (e.g. WASC, OWASP)
  - NIST SAMATE
  - want to understand tool capabilities
  - Secure code development
  - Secure coding standards
    - encourages the adoption of coding practices to avoid vulnerabilities (e.g. the CERT Secure Coding Standards project).
  - Language vulnerability avoidance
    - provide guidance to programmers in avoiding vulnerabilities inherent in programming languages and guidance to language developers in improving their language standards (e.g. ISO/IEC TR 24772 being developed by ISO/IEC JTC 1/SC 22/WG 3)
CWE Usage Modes & Scenarios

- Usage Modes:
  - Browse
  - Search
  - Lookup
  - Inspect

- Usage Scenarios:
  - Mapping
  - Compare
  - Learn More
  - Find Gaps
  - Find Related
  - Prioritize
  - Announce a Vulnerability

To subscribe, see: [http://cwe.mitre.org/community/registration.html](http://cwe.mitre.org/community/registration.html)

or just send an email to listserv@lists.mitre.org with the command: subscribe CWE-RESEARCH-LIST
Types of Views

- Lists: simple lists of CWE nodes for a specialized purpose.
- Organization Schemes: hierarchical or other organizational schemes that are for a specific purpose.

Views

V1 - Programming language-specific
- When programming or analyzing specific languages (C, Perl, Java, etc.), these are the issues of which you should be aware. Also, runtime vs. compiled, and other language-related characteristics.

V2 - Platform-specific
- When a program is run on a platform (Windows, UNIX, etc.) or in certain environments (32/64 bit, multi-processor), there are certain issues that should be checked for in addition to the actual language used. E.g., backslashes in paths, trailing filename dots, concurrency.

V3 - Technology-specific
- Is the weakness generic, or is it primarily associated with, or dependent on a certain technology class: Web, OS, Database?

V4 - Common Weakness Chains
- When viewing a weakness, it is useful to know related issues. The proper fix may not lie in the same place where the result is seen, so finding weakness they commonly lead to or result from a weakness is useful to support patching and visualize more abstract weakness relationships.

V5 - Taxonomy/Classification
- From a more formal taxonomic perspective, the most appropriate abstraction levels for various weaknesses may be important.
V6 - Commonality
- How easy is it for someone to make this mistake? How often is this weakness seen?

V7 - Risk/Severity-based
- Correlation by CWE to ensure that all "high" risk weaknesses have been addressed.

V8 - Feature-specific
- For a CWE, is it associated with other programming or security concepts? Does it usually involve or require features such as authentication, authorization, permissions, file access, or threading?

V9 - Resource-specific
- Is the weakness associated with a specific system resource such as memory, files, or network sockets?

V10 - Attack-based
- Typically, external researchers or auditors might perform testing on the running code. In this case, their results will most likely be described as attacks or vulnerabilities. If that is the case, a view supporting the CWEs grouped by the causal vulnerability and/or trigger attack may be useful.

V11 - Genesis
- A breakdown of issues based on which software development phase they typically occur in, e.g. design or implementation.

XS - CWE Cross-Section
- A small set of diverse CWE nodes that illustrates the breadth and depth of CWE.

SAMATE - SAMATE Slice
- The prioritized CWE nodes that are being focused on by SAMATE.

NVD - NVD Slice
- The set of CWE nodes that NVD will use to classify their entries.

SANS - SANS Secure Programming Information
- The set of CWE nodes that SANS' Secure Programming initiative is emphasizing for developer awareness.

OWASP - OWASP Top Ten
- The CWE nodes associated with the OWASP Top Ten.

Coverage of CWE

Draft

© 2007 MITRE
Types, Children, Categories & Parents

- Categories w/Children
- Categories w/o Children
- Types w/Children
- Types w/o Children

Node Types

- Grouping: arbitrary collection
- Category: general weakness type
- Variant: very low-level weakness type

Types

- Configuration
- Path Manipulation
- Equivalent Special Element Injection
- Type Errors
- Cross-Boundary Cleansing Information Leak
- Insecure Temporary File
- Byte/Object Code
- Inadvertent
- Relative Path Traversal
- Absolute Path Traversal
- Path Issue - trailing dot - `'filedir.'`
- Path Issue - internal dot - `'file.ordir'`
- Link Following
- Virtual Files
- Cross-site scripting (XSS)
- Argument Injection or Modification
- SQL injection
- CRLF Injection
- Direct Static Code Injection
- Out-of-bounds Read
- Trailing Special Element
- Internal Special Element
- Error Message Information Leaks
- Product-Generated Error Message Information Leak
- Password in Configuration File
- False Sensing
- Plaintext Storage of Sensitive Information
- Weak Encryption
- Insufficient Entropy
- Race condition within a thread
- J2EE Bad Practices: Threads
- Insufficient Resource Locking
- Unprotected Alternate Channel
- Untrusted Search Path
- Multiple Interpretation Error (MIE)
- Missing Initialization
- Web Parameter Tampering
- Insufficiently Protected Credentials
- Information Leak Through Caching
- Information Leak Through Log Files
- File and Directory Information Leaks
- Information Leak Through Source Code
- Missing Error Handling Mechanism
- Dead Code
- Incorrect Object Comparison: Syntactic
- Externally Controlled Reference to an Internal Resource

© 2007 MITRE
Node Types: Example

- **Issues that affect files**
  - Insufficient Input Validation
  - Path Equivalence
  - Directory Restriction
  - Path Traversal
  - XSS

- **Grouping:** arbitrary collection
- **Category:** general weakness type
- **Variant:** very low-level weakness type

Path Equivalence:
- Grouping: arbitrary collection
- Category: general weakness type
- Variant: very low-level weakness type

© 2007 MITRE
The “Mechanics” of a Vulnerability: Integer Overflow to Heap Overflow

Incorrect Range Check → Integer Overflow → Insufficient Memory Allocation → Heap Overflow → Code Execution

CWE-190: Integer Overflow
- Primary Weakness (Root Cause)

CWE-122: Heap Overflow
- Resultant Fault

Integer Overflow to Heap Overflow (chain)

Assumption: height and width are reasonable sizes.

```
A
size = height * width;
b surplus = malloc(size);
memmove(buf, InputBuf, SZ);
```

```
B
```
The buffer overflow occurs because the newly created buffer is smaller than expected, because the integer overflow causes the 'size' variable to be smaller than expected.
Relative Path Traversal (composite)

- Lack of "container" enforcement for file accesses (CWE-216, partial)
- 'Loose' Directory Permissions (CWE-275)
- Outsider influence of pathname (N/A, design-level)
- Interaction Error (null byte) (CWE-626)

• Application can potentially access anywhere user 'nobody' can
• No real built-in OS permissions for 'cannot navigate above this directory'
• Influence of pathname is typically a design-level decision, and can be done safely with proper pathname generation
• Null bytes widen the scope – 'cut off' .txt extension
Symbolic Link Following (composition)

Symbolic Link Following (composite)

CWE-61: Symlink Following
- Predictability: CWE-340
- Race Condition: CWE-362
- ‘Loose’ Directory Permissions: CWE-275
- Path Equivalence: CWE-41

- Filename can be predicted
- File can be created by other party before it is opened for writing
- File created in a shared directory with writable permissions
- Equivalence: a symlink can act as an alternate name for a critical file
Formalizing a Schema for Weaknesses

Identifying Information
- CWE ID
- Name

Describing Information
- Description
- Extended Description
- Alternate Terms
- Demonstrative Examples
- Observed Examples
- Context Notes
- Source Taxonomy
- References
- Whitebox Definition
- Blackbox Definition
- Formal Definition

Scoping & Delimiting Information
- Type
- 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
- Affected Resource
- Related Attacks
- Obscurity Vector
- Node Relationships
- Research Gaps

Available CWE resources

- Published CWE dictionary
- Additional MITRE materials
- Published references, incl. CVE, CERT
- Unpublished industry experience

1500 words in current descriptions
(16000 words with examples and notes)

30 entries vocabulary

Total 620 entries

Scope of the current project

© 2007 MITRE
CWE 122: Heap Overflow

Description: A heap overflow condition is a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of memory, generally meaning that the buffer was allocated using a routine such as the POSIX malloc() call.

Definition (Whitebox): Heap Overflow is a code path where the following must occur:

- dynamic allocation of a buffer
- data is written to the buffer where
  - the expected size of the buffer is greater than the actual size of the buffer where
    - expected size is equal to size of data added to position from which writing operation starts

Formal Definition

Heap Overflow is a buffer write operation such that a buffer of the buffer write operation is dynamically allocated and that the buffer write operation overflows the buffer.

where

Buffer write operation overflows buffer is a buffer write operation and that of a buffer such that the buffer write operation has the buffer and the size of the data item of the buffer write operation + the position of the buffer write operation > the size of the buffer.
Whitebox Definition of 3 CWEs

CWE 122: Heap Overflow
- Heap Overflow is a code path where the following must occur:
  - dynamic allocation of a buffer
  - data is written to the buffer where
    - the expected size of the buffer is greater than the actual size of the buffer where
      - expected size is equal to size of data added to position from which writing operation starts

CWE 457: Uninitialized Variable
- Uninitialized variable is a code path where the following must occur:
  - path starts with statement that defines variable
  - path ends with statement that access the variable
  - path does not contain a statement that assigns value to the variable

CWE 476: Null Dereference
- Null dereference is a code path where the following must occur:
  - path ends with a statement that dereferences a pointer
  - path starts with a statement that assigns a null value to the pointer
  - path does not contain a statement that assigns value to the pointer
OWASP Top Ten 2007

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

Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006
The National Vulnerability Database (NVD) tags Vulnerabilities with CWEs.

CWE is Now A Part of the NVD CVE Information

NVD Now Maps to CWE!

nvd.nist.gov