Software Security Knowledge: CWE

Knowing what could make software vulnerable to attack

Robert A. Martin
Sean Barnum
May 2011
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

<table>
<thead>
<tr>
<th>1. REPORT DATE</th>
<th>MAY 2011</th>
<th>2. REPORT TYPE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3. DATES COVERED</td>
<td>00-00-2011 to 00-00-2011</td>
</tr>
<tr>
<td>4. TITLE AND SUBTITLE</td>
<td></td>
<td>5a. CONTRACT NUMBER</td>
<td></td>
</tr>
<tr>
<td>Software Security Knowledge: CWE. Knowing what could make software vulnerable to attack</td>
<td></td>
<td>5b. GRANT NUMBER</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5c. PROGRAM ELEMENT NUMBER</td>
<td></td>
</tr>
<tr>
<td>6. AUTHOR(S)</td>
<td></td>
<td>5d. PROJECT NUMBER</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5e. TASK NUMBER</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5f. WORK UNIT NUMBER</td>
<td></td>
</tr>
<tr>
<td>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</td>
<td>Mitre Corp, 202 Burlington Road, Bedford, MA, 01730-1420</td>
<td>8. PERFORMING ORGANIZATION REPORT NUMBER</td>
<td></td>
</tr>
<tr>
<td>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. SPONSOR/MONITOR’S ACRONYM(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. SPONSOR/MONITOR’S REPORT NUMBER(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. DISTRIBUTION/AVAILABILITY STATEMENT</td>
<td>Approved for public release; distribution unlimited</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. SUPPLEMENTARY NOTES</td>
<td>Presented at the 23rd Systems and Software Technology Conference (SSTC), 16-19 May 2011, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. ABSTRACT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. SUBJECT TERMS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. SECURITY CLASSIFICATION OF:</td>
<td>a. REPORT unclassified</td>
<td>b. ABSTRACT unclassified</td>
<td>c. THIS PAGE unclassified</td>
</tr>
<tr>
<td>17. LIMITATION OF ABSTRACT</td>
<td>Same as Report (SAR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. NUMBER OF PAGES</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19a. NAME OF RESPONSIBLE PERSON</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agenda

8:00-8:45am Software Security Knowledge about Applications Weaknesses

9:00-9:45am Software Security Knowledge about Attack Patterns Against Applications
Training in Software Security

10:15-11:00am Software Security Practice

11:15-12:00am Supporting Capabilities
Assurance Cases
Secure Development & Secure Operations
Today Everything’s Connected

Your System is attackable...

When this Other System gets subverted through an un-patched vulnerability, a mis-configuration, or an application weakness...
The Software Supply Chain

If the weaknesses in software were as easy to spot and their impact as obvious as...
Vulnerability Type Trends:
A Look at the CVE List (2001 - 2007)
Removing and Preventing the Vulnerabilities Requires More Specific Definitions...CWEs

Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (79)
  • Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS) (80)
  • Improper Neutralization of Script in an Error Message Web Page (81)
  • Improper Neutralization of Script in Attributes of IMG Tags in a Web Page (82)
  • Improper Neutralization of Script in Attributes in a Web Page (83)
  • Improper Neutralization of Encoded URI Schemes in a Web Page (84)
  • Doubled Character XSS Manipulations (85)
  • Improper Neutralization of Invalid Characters in Identifiers in Web Pages (86)
  • Improper Neutralization of Alternate XSS Syntax (87)

Improper Restriction of Operations within the Bounds of a Memory Buffer (119)
  • Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (120)
  • Out-of-bounds Read (125)
  • Improper Handling of Length Parameter Inconsistency (130)
  • Improper Validation of Array Index (129)
  • Return of Pointer Value Outside of Expected Range (466)
  • Access of Memory Location Before Start of Buffer (786)
  • Access of Memory Location After End of Buffer (788)
  • Buffer Access with Incorrect Length Value (805)
  • Untrusted Pointer Dereference (822)
  • Use of Out-of-range Pointer Offset (823)
  • Access of Uninitialized Pointer (824)
  • Expired Pointer Dereference (825)

Path Traversal (22)
  • Relative Path Traversal (23)
    • Path Traversal: '../filedir' (24)
    • Path Traversal: '/..filedir' (25)
    • <--------------8 more here ------------->
    • Path Traversal: '..../' (34)
    • Path Traversal: '.../...//' (35)
  • Absolute Path Traversal (36)
    • Path Traversal: '/absolute/pathname/here' (37)
    • Path Traversal: 'absolute\pathname\here' (38)
    • Path Traversal: 'C:\dirname' (39)
    • Path Traversal: '\\UNC\share\name\' (Windows UNC Share) (40)
Exploitable Software Weaknesses (a.k.a. Vulnerabilities)

Vulnerabilities can be the outcome of non-secure practices and/or malicious intent of someone in the development/support lifecycle. The exploitation potential of a vulnerability is independent of the “intent” behind how it was introduced.

Intentional vulnerabilities are spyware & malicious logic deliberately imbedded (and might not be considered defects but they can make use of the same weakness patterns as unintentional mistakes)

Note: Chart is not to scale – notional representation -- for discussions
Common Weakness Enumeration (CWE)

- dictionary of weaknesses
  - weaknesses that can lead to exploitable vulnerabilities (i.e. CVEs)
  - the things we don’t want in our code, design, or architecture
  - web site with XML of content, sources of content, and process used
- structured views
  - provides multiple views into CWE dictionary content
  - supports alternate views – developer/researcher/sub-views
- open community process
  - to facilitate common terms/ concepts/facts and understanding
  - allows for vendors, developers, system owners and acquirers to understand tool capabilities/ coverage and priorities
  - utilize community expertise

Foundation for other DHS, NSA, OSD, NIST, OWASP, SANS, and OMG SwA Efforts

© 2011 MITRE
Building software only require a few skills and basic understanding...
...but sailing ships in the open ocean and building commerce and defense capabilities based upon them requires understanding...
...of navigation threats...
...surface maps didn’t capture the full set of threats and hazards – i.e. what was really going on...

...a more insightful depiction – one that shows what was going on under the surface – was needed...
...so “soundings” were made in important areas to identify and locate hidden hazards...
...and warning signals to help others avoid known hazards were erected along with...

...indicators showing safe ways to avoid the known hazards...
Know Security Weaknesses

Secure Coding Rules

Know Security Weaknesses

Know Security Weaknesses

Know Security Weaknesses

Know Security Weaknesses
But they also needed to deal with the people that were out there trying to locate vulnerabilities and weaknesses in their technologies, processes, or practices...
...with defensive and offensive security capabilities.
Security Feature

XSS (CWE-79) Attack (CAPEC-86)

SQL Injection (CWE-89) Attack (CAPEC-66)
“For years in computer security, we have been attempting to protect the broken stuff from the bad people by placing a barrier between the bad people and the broken stuff. We have failed. Instead, we need to fix the broken stuff so that attacking it successfully takes far more resources and skill than is currently the case.”
International in scope and free for public use, CWE™ provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design.

Building CWE & Consensus

Similar Standards
- Attack Patterns (CAPEC)
- Vulnerabilities (CWE)
- Configurations (CCE)
- Platforms (CPE)
- Malware (MAEC)
- Assessment Language (OVAL)
- Checklist Language (KCCDF)
- Log Format (CCE)
- Security Content Automation (SCAP)
- Making Security Measurable

© 2011 MITRE
PLOVER (CWE draft 1)

CWE draft 5

CWE draft 7

CWE Vers 1.0

CWE Vers 1.5

CWE Vers 1.12

<table>
<thead>
<tr>
<th>Year</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>300</td>
</tr>
<tr>
<td>2006</td>
<td>599</td>
</tr>
<tr>
<td>2007</td>
<td>634</td>
</tr>
<tr>
<td>2008</td>
<td>673</td>
</tr>
<tr>
<td>2009</td>
<td>799</td>
</tr>
<tr>
<td>Apr 2011</td>
<td>844</td>
</tr>
</tbody>
</table>
Gartner Magic Quadrant for Static Application Security Testing Tools

Plus Some Other Important Tool Players…

Cenzic
CAST Software
Polyspace
Security Innovation
LDRA
KDM Analytics
SureLogic
Programming Research Inc
SofCheck

Source: Gartner
As of December 2010

© 2011 MITRE
CWE Compatibility & Effectiveness Program
(launched Feb 2007)
cwe.mitre.org/compatible/
C/C++ “Breadth” Test Case Coverage

- No Tools: 42%
- One Tool: 12%
- Two Tools: 11%
- Three Tools: 13%
- Four Tools: 15%
- Five Tools: 7%

Java “Breadth” Test Case Coverage

- No Tools: 40%
- One Tool: 14%
- Two Tools: 15%
- Three Tools: 18%
- Four Tools: 12%
- Five Tools: 2%
- Six Tools: 0%

- PMD: 3%
- Coverity: 0%
- FindBugs: 1%
- Fortify: 7%
- Klocwork: 1%
- Ounce Labs: 3%
All of these are aimed at different aspects of understanding how well tools find CWEs in software applications and what can be done to improve that and standardize the process for expressing a tools capabilities.
### CWE Coverage – Implemented...

#### CWE IDs mapped to Klocwork Java issue types

From current

CWE IDs mapped to Klocwork Java issue types

See also Detected Java Issues.

<table>
<thead>
<tr>
<th>CWE ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>ABU/Tainted未検出込まれるバッファーオーバーフロー</td>
</tr>
<tr>
<td>22</td>
<td>SCH/Taintedгенерация случайных данных</td>
</tr>
<tr>
<td>73</td>
<td>SCH/TaintedCall INDEX_ACCESS</td>
</tr>
<tr>
<td>74</td>
<td>SCH/TaintedINJECTION</td>
</tr>
<tr>
<td>77</td>
<td>SCH/Code Injection</td>
</tr>
<tr>
<td>75</td>
<td>NNT/Tainted未検出ユーザーサーバーオーバーフロー</td>
</tr>
<tr>
<td>88</td>
<td>NNT/Insecure Script File in Web Server</td>
</tr>
</tbody>
</table>

#### 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

それ以外の情報. Detected C and C++ Issues.

<table>
<thead>
<tr>
<th>CWE ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>ABU/Tainted未検出込まれるバッファーオーバーフロー</td>
</tr>
<tr>
<td>22</td>
<td>SCH/Taintedгенерация случайных данных</td>
</tr>
<tr>
<td>73</td>
<td>SCH/TaintedCall INDEX_ACCESS</td>
</tr>
<tr>
<td>74</td>
<td>SCH/TaintedINJECTION</td>
</tr>
<tr>
<td>77</td>
<td>SCH/Code Injection</td>
</tr>
<tr>
<td>75</td>
<td>NNT/Tainted未検出ユーザーサーバーオーバーフロー</td>
</tr>
<tr>
<td>88</td>
<td>NNT/Insecure Script File in Web Server</td>
</tr>
</tbody>
</table>
With all of these CWEs, where do you start?
2009 SANS/ CWE Top 25 Programming Errors (released 12 Jan 2009)

- List selected by security experts from 34 organizations
2010 CWE/ SANS Top 25 Programming Errors
(released 16 Feb 2010)
cwe.mitre.org/top25/

- List selected by security experts from 34 organizations
Main Goals

- Raise awareness for developers
- Help universities to teach secure coding
- Empower customers who want to ask for more secure software
- Provide a starting point for in-house software shops to measure their own progress
Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

- CWE-20: Improper Input Validation
- CWE-116: Improper Encoding or Escaping of Output
- CWE-89: Failure to Preserve SQL Query Structure (aka 'SQL Injection')
- CWE-79: Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
- CWE-78: Failure to Preserve OS Command Structure (aka 'OS Command Injection')
- CWE-319: Cleartext Transmission of Sensitive Information
- CWE-352: Cross-Site Request Forgery (CSRF)
- CWE-362: Race Condition
- CWE-209: Error Message Information Leak

Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

- CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer
- CWE-642: External Control of Critical State Data
- CWE-73: External Control of File Name or Path
- CWE-426: Untrusted Search Path
- CWE-94: Failure to Control Generation of Code (aka 'Code Injection')
- CWE-494: Download of Code Without Integrity Check
- CWE-404: Improper Resource Shutdown or Release
- CWE-665: Improper Initialization
- CWE-682: Incorrect Calculation

Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

- CWE-327: Use of a Broken or Risky Cryptographic Algorithm
- CWE-259: Hard-Coded Password
- CWE-732: Insecure Permission Assignment for Critical Resource
- CWE-330: Use of Insufficiently Random Values
- CWE-250: Execution with Unnecessary Privileges
- CWE-602: Client-Side Enforcement of Server-Side Security
Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

For each weakness, its ranking in the general list is provided in square brackets.

<table>
<thead>
<tr>
<th>Rank</th>
<th>CWE ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>CWE-79</td>
<td>Failure to Preserve Web Page Structure ('Cross-site Scripting')</td>
</tr>
<tr>
<td>[2]</td>
<td>CWE-89</td>
<td>Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>[4]</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>[8]</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>[9]</td>
<td>CWE-78</td>
<td>Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>[17]</td>
<td>CWE-209</td>
<td>Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>[23]</td>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>[25]</td>
<td>CWE-362</td>
<td>Race Condition</td>
</tr>
</tbody>
</table>

Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

<table>
<thead>
<tr>
<th>Rank</th>
<th>CWE ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3]</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>[7]</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>[12]</td>
<td>CWE-805</td>
<td>Buffer Access with Incorrect Length Value</td>
</tr>
<tr>
<td>[13]</td>
<td>CWE-754</td>
<td>Improper Check for Unusual or Exceptional Conditions</td>
</tr>
<tr>
<td>[14]</td>
<td>CWE-98</td>
<td>Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')</td>
</tr>
<tr>
<td>[15]</td>
<td>CWE-129</td>
<td>Improper Validation of Array Index</td>
</tr>
<tr>
<td>[16]</td>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>[18]</td>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>[20]</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>[22]</td>
<td>CWE-770</td>
<td>Allocation of Resources Without Limits or Throttling</td>
</tr>
</tbody>
</table>

Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

<table>
<thead>
<tr>
<th>Rank</th>
<th>CWE ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6]</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>[10]</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>[24]</td>
<td>CWE-327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
</tbody>
</table>
2 **CWE-89: Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)**

**Summary**

<table>
<thead>
<tr>
<th>Weakness Prevalence</th>
<th>Consequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Data loss, Security bypass</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remediation Cost</th>
<th>Ease of Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Easy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attack Frequency</th>
<th>Attacker Awareness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Often</td>
<td>High</td>
</tr>
</tbody>
</table>

**Discussion**

These days, it seems as if software is all about the data: getting it into the database, pulling it from the database, massaging it into information, and sending it elsewhere for fun and profit. If attackers can influence the SQL that you use to communicate with your database, then suddenly all your fun and profit belongs to them. If you use SQL queries in security controls such as authentication, attackers could alter the logic of those queries to bypass security. They could modify the queries to steal, corrupt, or otherwise change your underlying data. They’ll even steal data one byte at a time if they have to, and they have the patience and know-how to do so.

**Prevention and Mitigations**

**Architecture and Design**

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using persistence layers such as Hibernate or Enterprise Java Beans, which can provide significant protection against SQL injection if used properly.

**Detection Methods**

I If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Process SQL queries using prepared statements, parameterized queries, or stored procedures. These features should accept parameters or variables and support strong typing. Do not dynamically construct and execute query strings within these features using "exec" or similar functionality, since you may...
These mitigations will be effective in eliminating or reducing the severity of the Top 25. These mitigations will also address many weaknesses that are not even on the Top 25. If you adopt these mitigations, you are well on your way to making more secure software.

A Monster Mitigation Matrix is also available to show how these mitigations apply to weaknesses in the Top 25.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Establish and maintain control over all of your inputs.</td>
</tr>
<tr>
<td>M2</td>
<td>Establish and maintain control over all of your outputs.</td>
</tr>
<tr>
<td>M3</td>
<td>Lock down your environment.</td>
</tr>
<tr>
<td>M4</td>
<td>Assume that external components can be subverted, and your code can be read by anyone.</td>
</tr>
<tr>
<td>M5</td>
<td>Use industry-accepted security features instead of inventing your own.</td>
</tr>
<tr>
<td>GP1</td>
<td>(general) Use libraries and frameworks that make it easier to avoid introducing weaknesses.</td>
</tr>
<tr>
<td>GP2</td>
<td>(general) Integrate security into the entire software development lifecycle.</td>
</tr>
<tr>
<td>GP3</td>
<td>(general) Use a broad mix of methods to comprehensively find and prevent weaknesses.</td>
</tr>
<tr>
<td>GP4</td>
<td>(general) Allow locked-down clients to interact with your software.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>CWE</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-22</strong>: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>Mod</td>
<td>High</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-78</strong>: Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>Mod</td>
<td>High</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-79</strong>: Failure to Preserve Web Page Structure ('Cross-site Scripting')</td>
</tr>
<tr>
<td>Mod</td>
<td>High</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-89</strong>: Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>Mod</td>
<td>High</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-98</strong>: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-120</strong>: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>High</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-129</strong>: Improper Validation of Array Index</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-131</strong>: Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-190</strong>: Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-209</strong>: Information Exposure Through an Error Message</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-235</strong>: Improper Access Control (Authorization)</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-306</strong>: Missing Authentication for Critical Function</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-311</strong>: Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-322</strong>: Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-352</strong>: Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-362</strong>: Race Condition</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-434</strong>: Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-494</strong>: Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-601</strong>: URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-754</strong>: Improper Check for Unusual or Exceptional Conditions</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-770</strong>: Allocation of Resources Without Limits or Throttling</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-798</strong>: Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-805</strong>: Buffer Access with Incorrect Length Value</td>
</tr>
<tr>
<td>Mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>CWE-802</strong>: Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
</tbody>
</table>
### Focus Profiles

The prioritization of items in the general Top 25 list is just that - general. The rankings, and even the selection of which items should be included, can vary widely depending on context. Ideally, each organization can decide how to rank weaknesses based on its own criteria, instead of relying on a single general-purpose list.

A separate document provides several "focus profiles" with their own criteria for selection and ranking, which may be more useful than the general list.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>On the Cusp:</strong> Weakenes that Did Not Make the 2010 Top 25</td>
<td>From the original nominee list of 41 submitted CWE entries, the Top 25 was selected. This &quot;On the Cusp&quot; profile includes the remaining 16 weaknesses that did not make it into the final Top 25.</td>
</tr>
<tr>
<td>Educational Emphasis</td>
<td>This profile ranks weaknesses that are important from an educational perspective within a school or university context. It focuses on the CWE entries that graduating students should know, including historically important weaknesses.</td>
</tr>
<tr>
<td>Weaknesses by Language</td>
<td>This profile specifies which weaknesses appear in which programming languages. Notice that most weaknesses are actually language-independent, although they may be more prevalent in one language or another.</td>
</tr>
<tr>
<td>Weaknesses Typically Fixed in Design or Implementation</td>
<td>This profile lists weaknesses that are typically fixed in design or implementation.</td>
</tr>
<tr>
<td>Automated vs. Manual Analysis</td>
<td>This profile highlights which weaknesses can be detected using automated versus manual analysis. Currently, there is very little public, authoritative information about the efficacy of these methods and their utility. There are many competing opinions, even among experts. As a result, these ratings should only be treated as guidelines, not rules.</td>
</tr>
<tr>
<td>Weaknesses by Language</td>
<td>This profile specifies which weaknesses appear in which programming languages. Notice that most weaknesses are actually language-independent, although they may be more prevalent in one language or another.</td>
</tr>
<tr>
<td>For Developers with Established Software Security Practices</td>
<td>This profile is for developers who have already established security in their practice. It uses votes from the major developers who contributed to the Top 25.</td>
</tr>
<tr>
<td>Ranked by Importance - for Software Customers</td>
<td>This profile ranks weaknesses based primarily on their importance, as determined from the base voting data that was used to create the general list. Prevalence is included in the scores, but it has much less weighting than importance.</td>
</tr>
<tr>
<td>Weaknesses by Technical Impact</td>
<td>This profile lists weaknesses based on their technical impact, i.e., what an attacker can accomplish by exploiting each weakness.</td>
</tr>
</tbody>
</table>
Background Details to Check Out

- Process description
- Changelog for each revision
- On the Cusp – weaknesses that almost made it
- Appendices
  - Selection Criteria and Supporting Fields
  - Threat Model for the Skilled, Determined Attacker

[cwe.mitre.org/top25]
Frequently Asked Questions (FAQ)

How is this different from the OWASP Top Ten?
The short answer is that the OWASP Top Ten covers more general concepts and is focused on web applications. The CWE Top 25 covers a broader range of issues than what arise from the web-centric view of the OWASP Top Ten, such as buffer overflows. Also, one goal of the CWE Top 25 is to be at a level that is directly actionable to programmers, so it contains more detailed issues than the categories being used in the Top Ten. There is some overlap, however, since web applications are so prevalent, and some issues in the Top Ten have general applications to all classes of software.

How are the weaknesses prioritized on the list?
With the exception of Input Validation being listed as number 1 (partially for educational purposes), there is no concrete prioritization. Prioritization differs widely depending on the audience (e.g. web application developers versus OS developers) and the risk tolerance (whether code execution, data theft, or denial of service are more important). It was also believed that the use of categories would help the organization of the document, and prioritization would impose a different ordering.

Why are you including overlapping concepts like input validation and XSS, or incorrect calculation and buffer overflows? Why do you have mixed levels of abstraction?
While it would have been ideal to have a fixed level of abstraction and no overlap between weaknesses, there are several reasons why this was not achieved.

Contributors sometimes suggested different CWE identifiers that were closely related. In some cases, this difference was addressed by using a more abstract CWE identifier that covered the relevant cases.

In other situations, there was strong advocacy for including lower-level issues such as SQL injection and cross-site scripting, so these were added. The general trend, however, was to use more abstract weakness types.

While it might be desired to minimize overlap in the Top 25, many vulnerabilities actually deal with the interaction of 2 or more weaknesses. For example, external control of user state data (CWE-642) could be an important weakness that enables cross-site scripting (CWE-79) and SQL injection (CWE-89). To eliminate overlap in the Top 25 would lose some of this important subtlety.

Finally, it was a conscious decision that if there was enough prevalence and severity, design-related weaknesses would be included. These are often thought of as being more abstract than weaknesses that arise during implementation.

The Top 25 list tries to strike a delicate balance between usability and relevance, and we believe that it does so, even with this apparent imperfection.

Why don't you use hard statistics to back up your claims?
The appropriate statistics simply aren't publicly available. The publicly available statistics are either too high-level or not comprehensive enough. And none of them are comprehensive across all software types and environments.
People are Starved for Simplicity
The Top 25 is not…

- A silver bullet
- A guarantee of software health
- A perfect match for your unique needs
- As simple as it seems
- The only thing to include in contract language
- Completely found by tools
The Top 25 is...

- A mechanism for awareness
- A trigger of questions
- A place for mitigations
- A conversation starter
- A first step on the long road to software assurance
CWE Top 25 for 2011

- Started last month
- Utilizing the Common Weakness Scoring System (CWSS 0.4) and the Common Weakness Risk Assessment Framework (CWRAF 0.4) as under-pinning
- Will have numerous “Top 10’s” & one “Top 25”
  - Including Web, Embedded, e-Voting,…
- Final "master" Top 25 list, will leverage combined score from multiple vignettes.
- No fixed date for release of the 2011 Top 25 at this point, may take 2 to 3 months.
Common Weakness Scoring System (CWSS)

Archetypes:
- Web Browser User Interface
- Web Servers
- Application Servers
- Database Systems
- Desktop Systems
- SSL

Vignettes:
1. Web-based Retail Provider
2. Intranet resident health records management system of hospital
Business Value Context (BVC)

- Identifies critical assets and security concerns
- Links Technical Impacts (derived from CWE weaknesses) with business implications
- More fine-grained model than the CIA Triad

CWE Technical Impacts

1. Modify memory
2. Read memory
3. Modify files or directories
4. Read files or directories
5. Modify application data
6. Read application data
7. DoS: crash / exit / restart
8. DoS: amplification
9. DoS: instability
10. DoS: resource consumption (CPU)
11. DoS: resource consumption (memory)
12. DoS: resource consumption (other)
13. Execute unauthorized code or commands
14. Gain privileges / assume identity
15. Bypass protection mechanism
16. Hide activities
Calculating CWSS Impact Weights

**Technical Impact Scorecard**

1. Execute System Code: 10 - 6
2. Read System Data: 6
3. System Unstable Execution: 3
4. Network Resource consumption: 2
5. Read Application Data: 1

CWE-x:
- Execute System Code
- Network Resource Consumption
- Max (10, 2) / 10.0
- 1.0

CWE-y:
- System Unstable Execution
- Read System Data
- Max (3, 6) / 10.0
- 0.6

CWE-z:
- Read Application Data
- Max (1) / 10.0
- 0.1
Common Weakness Scoring System (CWSS)

Archetypes:
- Web Browser User Interface
- Web Servers
- Application Servers
- Database Systems
- Desktop Systems
- SSL

Vignettes:
1. Web-based Retail Provider
2. Intranet resident health records management system of hospital
Scoring Weaknesses Discovered in Code using CWSS

Steps:
1. Establish weightings for the vignette
2. Run code through analysis tool(s)
3. Tools produce report of CWE’s found in code
4. CWSS scoring engine automatically scores each CWE based on vignette definition
5. Go to step 2 for each piece of code applicable to this vignette

Line 23: CWE-109
Line 72: CWE-84
Line 104: CWE-482
Line 212: CWE-9
Line 213: CWE-754
...

Line 212: CWE-9: 9.9
Line 72: CWE-84: 7.9
Line 23: CWE-109: 5.6
Line 104: CWE-482: 3.1
Line 213: CWE-754: 0.0
...

Step 1 is only done once – the rest is automatic
Scoring Relevant Weaknesses using CWSS

Steps:
1. Establish weightings for the vignette
2. CWSS scoring engine processes each relevant CWE entry and automatically scores the entry based on vignette definition
3. CWE entries presented in priority order based on vignette-driven CWSS scores
4. Organization now has its own customized “Top N list” of critical weaknesses for this vignette

Step 1 is only done once – the rest is automatic
CWSS for a Technology Group

50% Web Vignette 1 ... TI(1), TI(2), TI(3),... Top N List 1
10% Web Vignette 2 ... TI(1), TI(2), TI(3),... Top N List 2
10% Web Vignette 3 ... TI(1), TI(2), TI(3),... Top N List 3
10% Web Vignette 4 ... TI(1), TI(2), TI(3),... Top N List 4
15% Web Vignette 5 ... TI(1), TI(2), TI(3),... Top N List 5
15% Web Vignette 6 ... TI(1), TI(2), TI(3),... Top N List 6

Web Application Technology Group

Web Application Technology Group

CWE Top 10 List for Web Applications can be used to:

- Identify skill and training needs for your web team
- Include in T’s & C’s for contracting for web development
- Identify tool capability needs to support web assessment
<table>
<thead>
<tr>
<th>Technology Group</th>
<th>Archetypes/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Applications</td>
<td>Web browser, web-server, web-based applications and services, etc.</td>
</tr>
<tr>
<td>Industrial Control Systems</td>
<td>SCADA, process control system, etc.</td>
</tr>
<tr>
<td>Real-time, Embedded Systems</td>
<td>Embedded Device, Programmable logic controller, implanted medical devices, avionics package.</td>
</tr>
<tr>
<td>End-point Computing Devices</td>
<td>Smart phone, laptop, personal digital assistant (PDA), and other remote devices that leave the enterprise and/or connect remotely to the enterprise.</td>
</tr>
<tr>
<td>Cloud Computing</td>
<td>Hosted applications or capabilities provided over the Internet, including Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure as a Service (IaaS).</td>
</tr>
<tr>
<td>Operating Systems</td>
<td>General-purpose OS, virtualized OS, Real-time operating system (RTOS), hypervisor, microkernel.</td>
</tr>
<tr>
<td>Enterprise Desktop Applications/Systems</td>
<td>Office products such as word processing, spreadsheets, project management, etc.</td>
</tr>
<tr>
<td>Domain Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E-Commerce</td>
<td>The use of the Internet or other computer networks for the sale of products and services, typically using on-line capabilities.</td>
</tr>
<tr>
<td>Banking &amp; Finance</td>
<td>Financial services, including banks, stock exchanges, brokers, investment companies, financial advisors, and government regulatory agencies.</td>
</tr>
<tr>
<td>Public Health</td>
<td>Health care, medical encoding and billing, patient information/data, critical or emergency care, medical devices (implantable, partially embedded, patient care), drug development and distribution, food processing, clean water treatment and distribution (including dams and processing facilities), etc.</td>
</tr>
<tr>
<td>Energy</td>
<td>Smart Grid (electrical network through a large region, using digital technology for monitoring or control), nuclear power stations, oil and gas transmission, etc.</td>
</tr>
<tr>
<td>Chemical</td>
<td>Chemical processing and distribution, etc.</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Plants and distribution channels, supply chain, etc.</td>
</tr>
<tr>
<td>Shipping &amp; Transportation</td>
<td>Aerospace systems (such as safety-critical ground aviation systems, on-board avionics, etc), shipping systems, rail systems, etc.</td>
</tr>
<tr>
<td>National Security</td>
<td>National security systems (including networks and weapon systems), Defense Industrial Base, etc.</td>
</tr>
<tr>
<td>Government and Commercial Security</td>
<td>Homeland Security systems, commercial security systems, etc.</td>
</tr>
<tr>
<td>Emergency Services</td>
<td>Systems and services that support first responders, incident management and response, law enforcement, and emergency services for citizens, etc.</td>
</tr>
<tr>
<td>Telecommunications</td>
<td>Cellular services, land lines, VOIP, cable &amp; fiber networks, etc.</td>
</tr>
<tr>
<td>Telecommuting &amp; Teleworking</td>
<td>Support for employees to have remote access to internal business networks and capabilities.</td>
</tr>
<tr>
<td>eVoting</td>
<td>Electronic voting systems, as used within state-run elections, shareholder meetings, etc.</td>
</tr>
</tbody>
</table>
Vignettes – Technology Groups & Business/Mission Domains

<table>
<thead>
<tr>
<th>Technology Groups</th>
<th>Business/Mission Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Applications</td>
<td>e-Commerce</td>
</tr>
<tr>
<td></td>
<td>Banking &amp; Finance</td>
</tr>
<tr>
<td></td>
<td>Energy (i.e., SmartGrid,</td>
</tr>
<tr>
<td></td>
<td>oil/gas transmission)</td>
</tr>
<tr>
<td></td>
<td>Chemical</td>
</tr>
<tr>
<td></td>
<td>Manufacturing</td>
</tr>
<tr>
<td></td>
<td>Shipping &amp; Transportation (i.e., rail, post)</td>
</tr>
<tr>
<td></td>
<td>Freight, ships, airlines, aerospace, postal</td>
</tr>
<tr>
<td></td>
<td>National Defense (e.g., national security)</td>
</tr>
<tr>
<td></td>
<td>Homeland Security (e.g., Homeland security agencies)</td>
</tr>
<tr>
<td></td>
<td>Secret Service (e.g., Secret Service)</td>
</tr>
<tr>
<td></td>
<td>Government (other than Nat’l Def &amp; HS)</td>
</tr>
<tr>
<td></td>
<td>Emergency Services (e.g., law enforcement)</td>
</tr>
<tr>
<td></td>
<td>Public Health</td>
</tr>
<tr>
<td></td>
<td>Food &amp; Water</td>
</tr>
<tr>
<td></td>
<td>Telecommunications</td>
</tr>
<tr>
<td></td>
<td>Teleworking</td>
</tr>
<tr>
<td></td>
<td>e-Voting</td>
</tr>
</tbody>
</table>

Vignettes with Archetypes to identify top CWEs in respective Domain/Technology Groups.
CWRAF: Common Weakness Risk Analysis Framework

**Domains**
- Chemical
- Energy
- E-Voting

**Technology Groups**
- Control Systems
- Web Applications
- Embedded Devices

**Vignettes**
- SCADA
- HMI
- Smart Grid
- House Meter

**Business Value Context**
- Financial loss, privacy violation

**Technical Impact Scorecard**
- 10 - Code execution
- 6 - Read Sensitive Data
- 3 - DoS: Unpredictable Execution
Customizing CWRAF to a Single In-house Software Package

**Domains**
- B2B Communication
- Admin and Maintenance
- Public E-Commerce Site

**Technology Groups**
- Databases / Storage
- Web Applications
- Mobile Apps

**Vignettes**
- Payment
- Admin Console
- Product Search / Browse
- Shopping Cart

**Business Value Context**
- Financial loss, privacy violation

**Technical Impact Scorecard**
- 10 - Code execution
- 6 - Read Sensitive Data
- 3 - DoS: Unpredictable Execution

**Context**
- Code execution (6)
- Read Sensitive Data (3)
- DoS: Unpredictable Execution (10)
Relationships between CWRAF, CWSS, and CWE

- **CWRAF**: Provides Vignettes (technical & business context) to specify relevant, applicable CWE IDs
- **CWEs (by ID)**: CWE 79, CWE 120, CWE 78, CWE 352, CWE 434, CWE 22, CWE 89, CWE 285, CWE 311, CWE 807

**CWSS**
- Influences Scoring Using Business Value and Technical Context
- Applies Scoring Criteria to Rank Relevant Weaknesses

**Note**: CWSS can be used in the context of CWRAF; but it is not a requirement.

- **Vignette A**: CWE 22, CWE 89
- **Vignette B**: CWE 79, CWE 89
- **Vignette C**: CWE 120, CWE 311

Provides results in prioritized lists of relevant CWE IDs for specific Vignettes.
Questions?