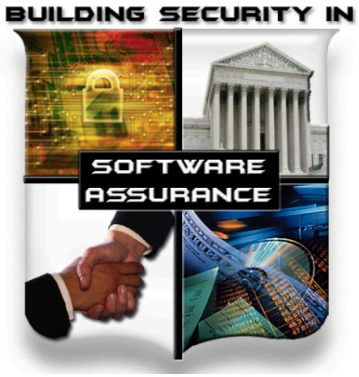




# Measuring Software Security

## Moving Towards Software Assurance Automation

## Part 1

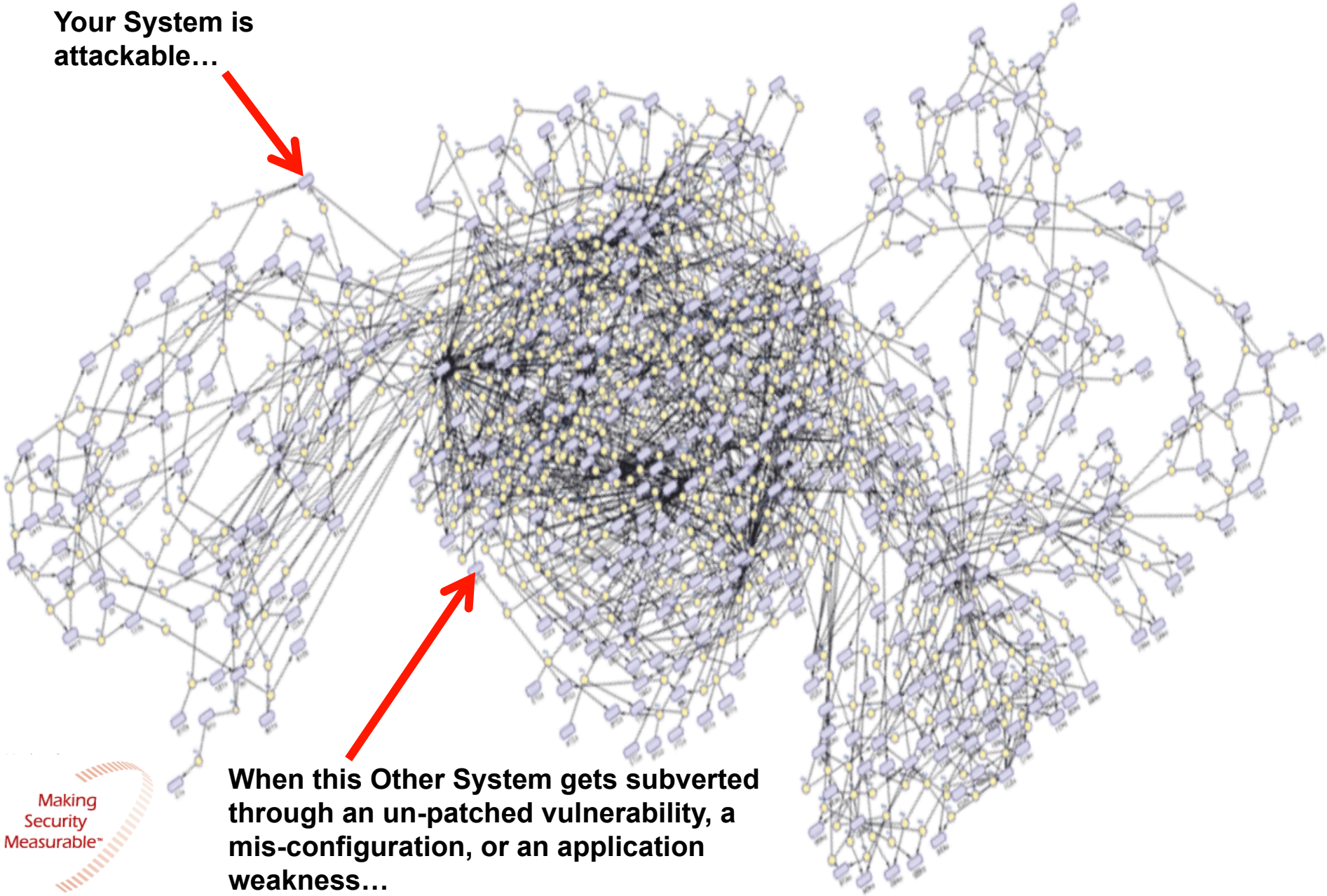
**Joe Jarzombek, DHS Director, Software Assurance**
**Robert Martin, MITRE CWE Project Lead**

**August 21, 2012**


Homeland Security

MITRE

# Today Everything's Connected – Like an Ecosystem




Making Security Measurable™

# SOFTWARE ASSURANCE FORUM

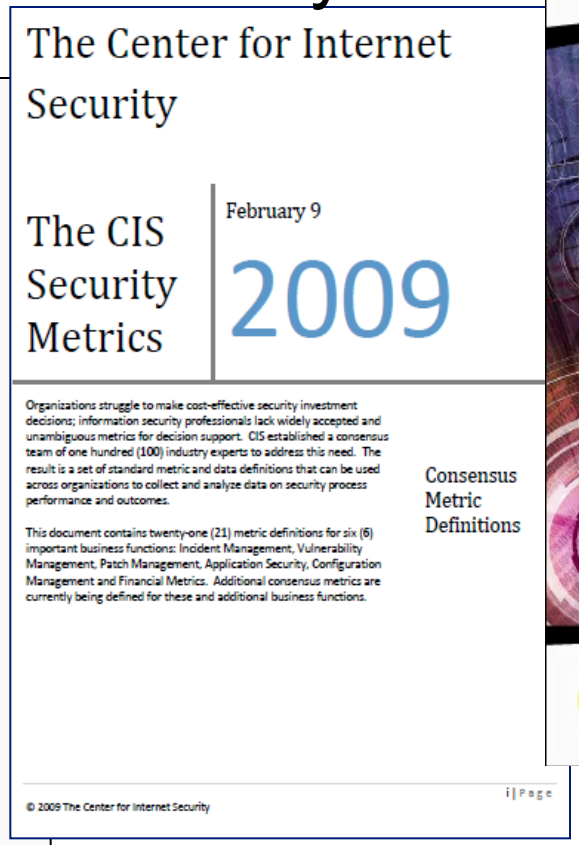
## BUILDING SECURITY IN

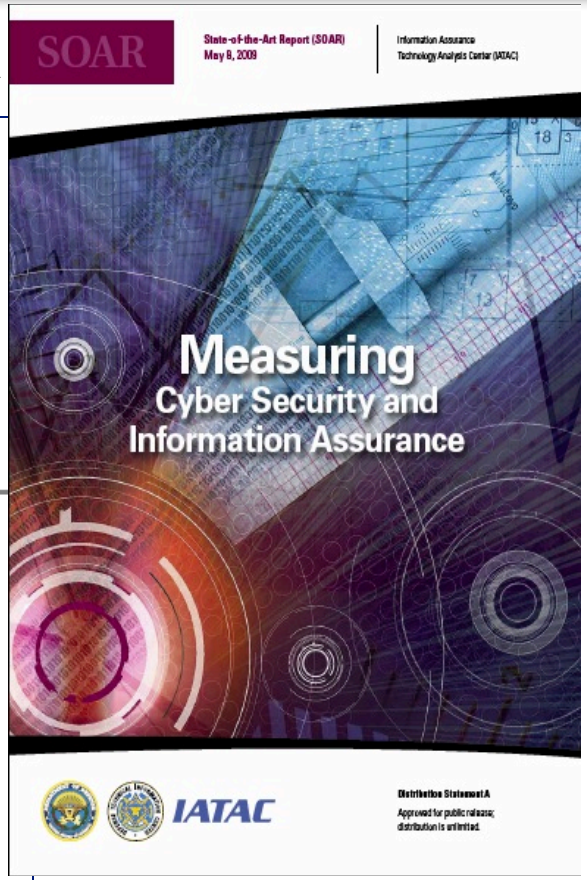
### *Security Measurement Resources*

Oct 08 → Feb 09 → May 09 →

**Practical Measurement Framework for Software Assurance and Information Security**

**Oct 2008**


**Exploitable Software Weaknesses are sources for future Zero-Day Attacks**

# Security is a Requisite Quality Attribute:
## Vulnerable Software Enables Exploitation

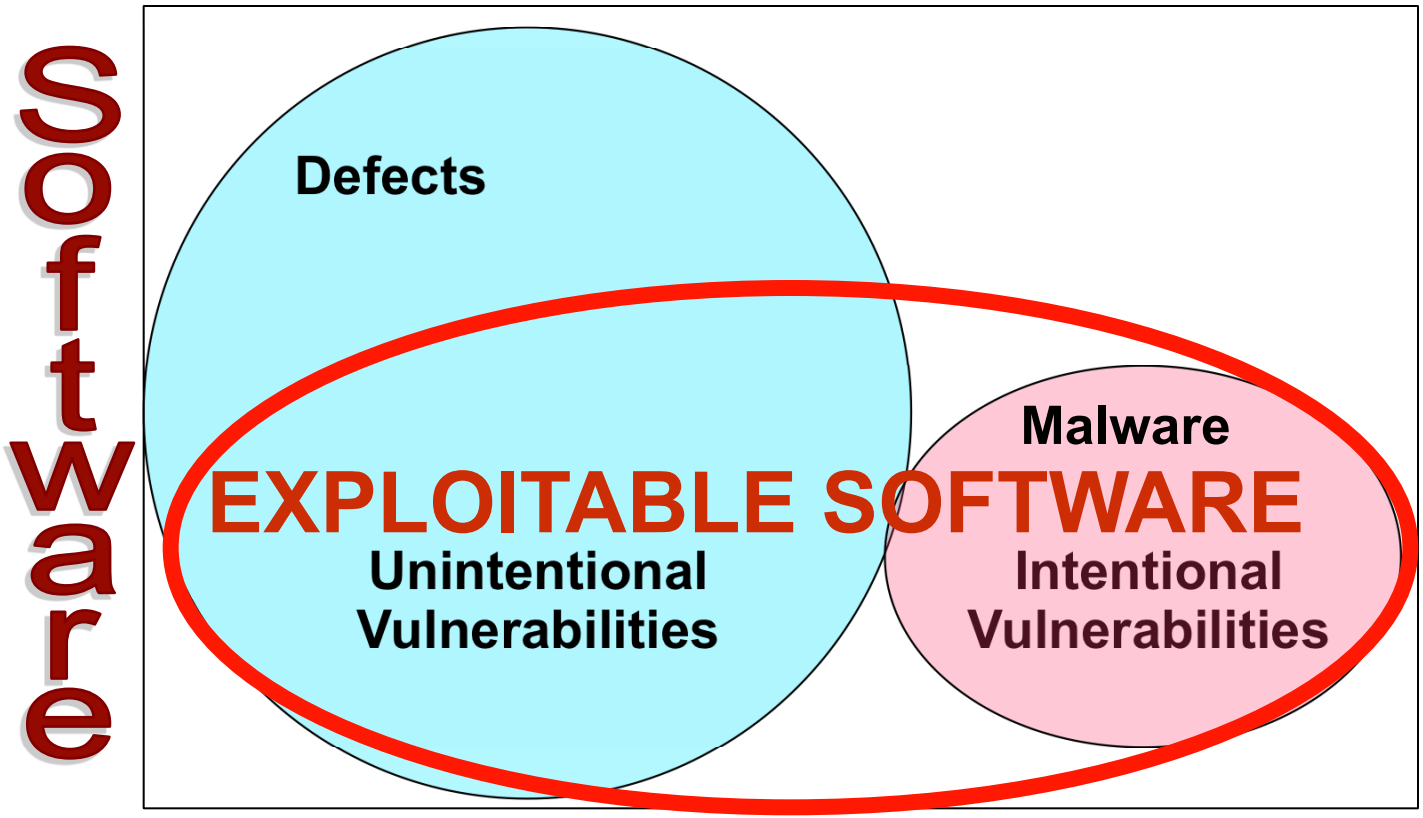
- Rather than attempt to break or defeat network or system security, hackers are opting to target application software to circumvent security controls.
  - **75% of hacks occurred at application level**
    - "90% of software attacks were aimed at application layer" (Gartner & Symantec, June 2006)
  - most exploitable software vulnerabilities are attributable to non-secure coding practices (and not identified in testing).
- Functional correctness must be exhibited even when software is subjected to abnormal and hostile conditions


In an era riddled with asymmetric cyber attacks, claims about system reliability, integrity & safety must include provisions for built-in security of the enabling software.

Homeland Security

# Software Assurance Addresses Exploitable Software:
Outcomes of non-secure practices and/or malicious intent

**Exploitation potential of vulnerability is independent of "intent"**




**'High quality' can reduce security flaws attributable to defects; yet traditional S/W quality assurance does not address intentional malicious behavior in software**

*Intentional vulnerabilities: spyware & malicious logic deliberately imbedded (might not be considered defects)

**Software Assurance (SwA) is the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software throughout the life cycle.***

***From CNSS Instruction 4009 "National Information Assurance Glossary" (26APR2010)***

# Software Security Assurance: Not just a good idea

- Many people responsible for protecting most critical infrastructure facilities have felt comfortable about security of their systems.
  - Facilities rely on industrial control systems (ICS) -- custom-built suites of systems that control essential mechanical functions of power grids, processing plants, etc -- usually not connected to the Internet, also known as "air-gapped."
  - Many industry owners, operators and regulators believed that this security model provided an infallible, invulnerable barrier to malicious cyber attacks from criminals and advanced persistent threat (APT) adversaries.
- National Defense Authorization Act (NDAA) -- which included a focus on software security (in Section 932, Strategy on Computer Software Assurance) -- serves as first cybersecurity law of 2011 and requires the U.S. Dept of Defense to develop a strategy for ensuring the security of software applications.
- Software Security Assurance, a set of practices for ensuring proactive application security, is key to making applications compliant with this new law.

**"How Stuxnet Demonstrates That Software Assurance Equals Mission Assurance:**
The rules of the game have changed," by Rob Roy, Federal CTO of Fortify, an HP Company

# Software Security Assurance: Not just a good idea

Steps organizations can take now to support software security assurance.

Tips from white paper on "7 Practical Steps to Delivering More Secure Software":

1. Quickly evaluate current state of software security and create a plan for dealing with it throughout the life cycle.
2. Specify the risks and threats to the software so they can be eliminated before they are deployed.
3. Review the code for security vulnerabilities introduced during development.
4. Test and verify the code for vulnerabilities.
5. Build a gate to prevent applications with vulnerabilities from going into production.
6. Measure the success of the security plan so that the process can be continually improved.
7. Educate stakeholders about security so they can implement the security plan.

Any development organization can implement this security plan immediately and begin to receive a return on their efforts within a minimal period of time. The key is to start now.

To complement the software strategy, there are several other areas of good security practices to observe and implement if they are not already part of the organizational security approach:

1. Implement software configurations such as the U.S. Government Configuration Baseline (formerly the Federal Desktop Core Configuration), strong authentication, and strict, documented internal policies and procedures.
2. Ask vendors to provide guarantees of software security as required by HR 6523.
3. Insert and enforce software assurance requirements in contracts.
4. Review IT security policies to ensure that all users of organizational networks and data comply with the strictest security policies possible with respect to the mission.
5. Determine how much risk the organization can afford and who is accountable for that risk. Constructing a new building in parts of California without accounting for earthquakes is unacceptable.

**"How Stuxnet Demonstrates That Software Assurance Equals Mission Assurance:**
The rules of the game have changed," by Rob Roy, Federal CTO of Fortify, an HP Company
http://email.tailorednews.com/r/jm892fwx7ega4ZTy4QI.htm

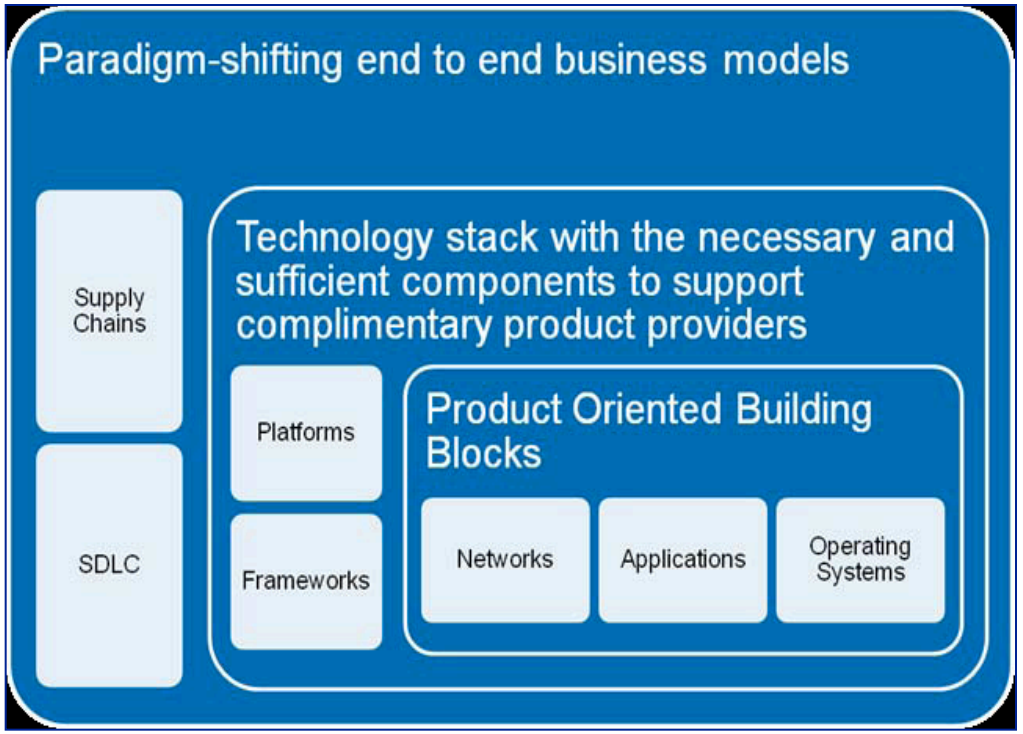
Building software without accounting for security is no longer an acceptable risk.

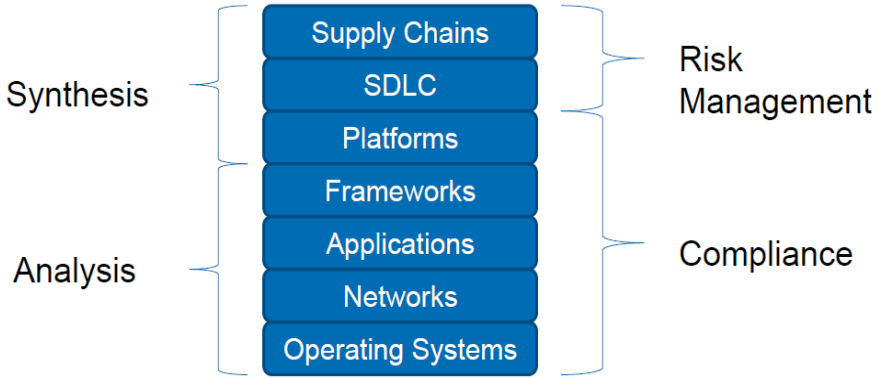
# IT/software security risk landscape is a convergence between “defense in depth” and “defense in breadth”

Enterprise Risk Management and Governance are security motivators

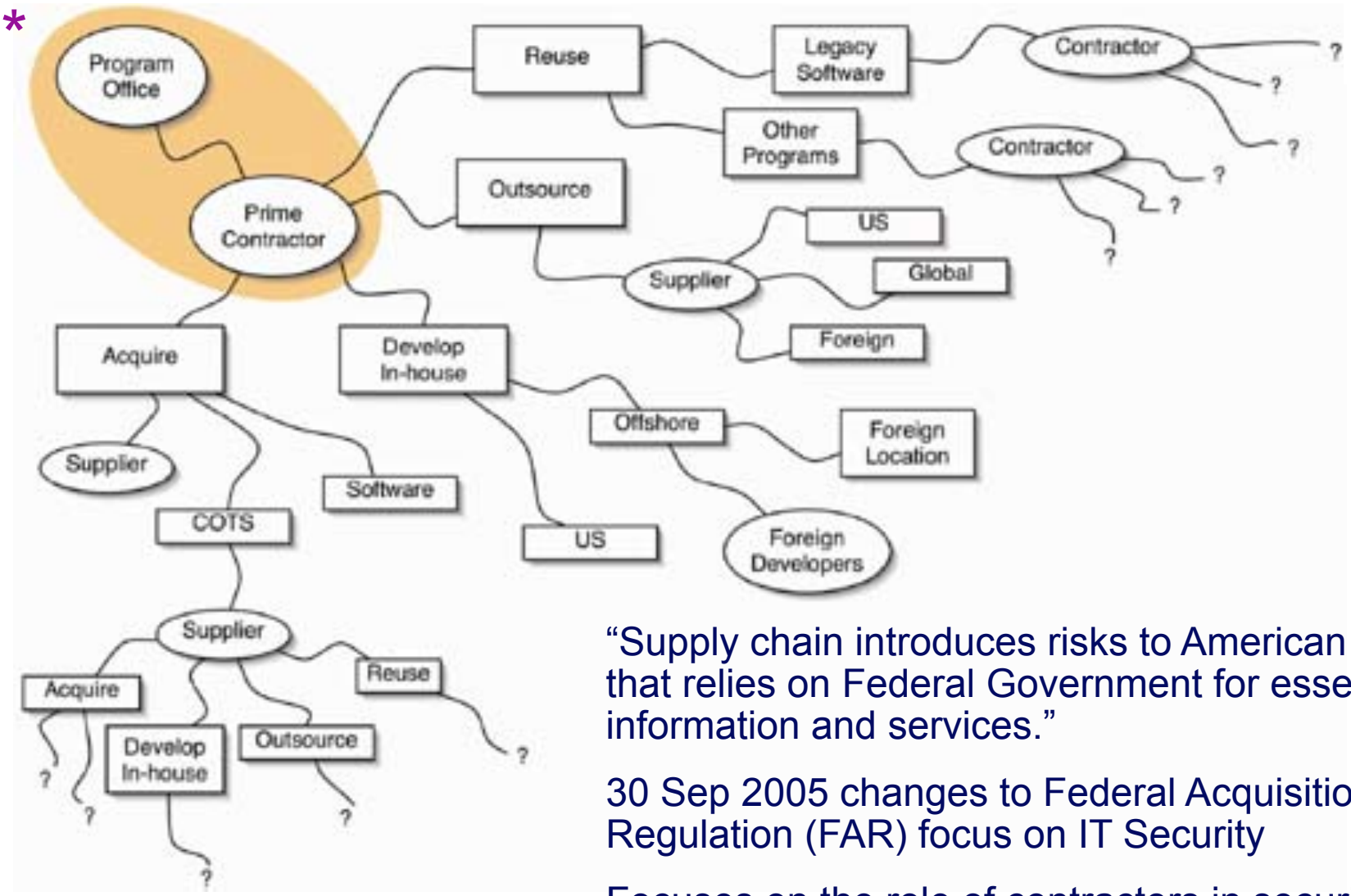
Acquisition could be considered the beginning of the lifecycle; more than development

> “In the digital age, sovereignty is demarcated not by territorial frontiers but by supply chains.”
>
> – Dan Geer, CISO In-Q-Tel




Software Assurance provides a focus for:
-- Secure Software Components,
-- Security in the Software Life Cycle,
-- Software Security in Services, and
-- Software Supply Chain Risk Management




*

"Supply chain introduces risks to American society that relies on Federal Government for essential information and services."

30 Sep 2005 changes to Federal Acquisition Regulation (FAR) focus on IT Security

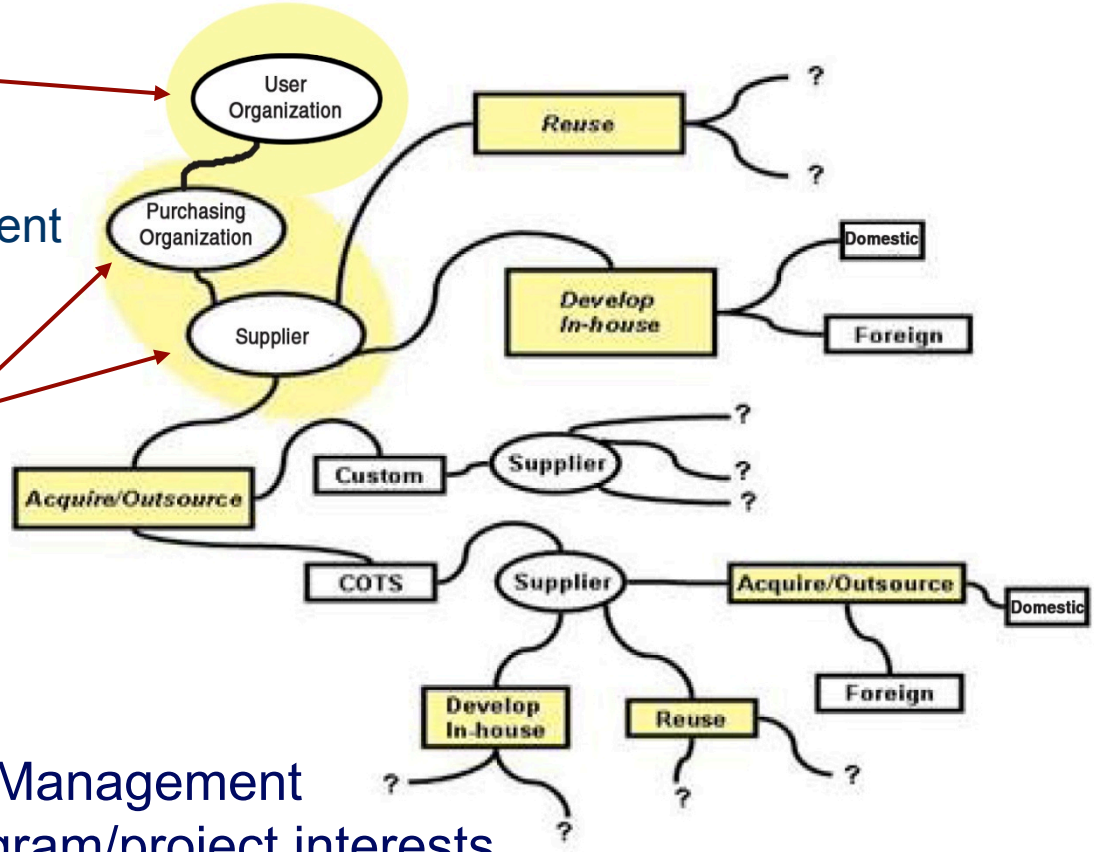
Focuses on the role of contractors in security as Federal agencies outsource various IT functions.

Homeland Security

"Scope of Supplier Expansion and Foreign Involvement" graphic in DACS www.softwaretechnews.com Secure Software Engineering, July 2005 article "Software Development Security: A Risk Management Perspective" synopsis of May 2004 GAO-04-678 report "Defense Acquisition: Knowledge of Software Suppliers Needed to Manage Risks"

# Risk Management (Enterprise ⟷ Project):
## Shared Processes & Practices ⟷ Different Focuses

- Enterprise-Level:
  - Regulatory compliance
  - Changing threat environment
  - Business Case
- Program/Project-Level:
  - Cost
  - Schedule
  - Performance




Software Supply Chain Risk Management traverses enterprise and program/project interests

1. Insert and enforce software assurance requirements in contracts.
2. Review IT security policies to ensure that all users of organizational networks and data comply with the strictest security policies possible with respect to the mission.
3. Determine how much risk the organization can afford and who is accountable for that risk.

Homeland Security

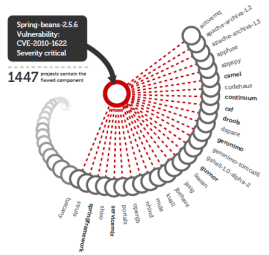
# Thousands of downloads from open libraries with documented vulnerabilities

## Security Vulnerabilities

It's not uncommon for vulnerabilities to be discovered in popular components. Updates that address the problem are typically provided quickly. However, even when security warnings are posted and easily accessible, they are often overlooked. In March 2009, the United States Computer Emergency Readiness Team and the National Institute of Standards and Technology (US-CERT/NIST) issued a warning that the Legion of the Bouncy Castle Java Cryptography API component was extremely vulnerable to remote attacks. In January 2011, almost 2 years later, 1,651 different organizations downloaded the vulnerable version of Bouncy Castle from the Central Repository within a single month. [v] In January 2010, the US-CERT/NIST posted an alert via their National Vulnerability Database that Jetty had a critical security flaw, which might allow attackers to execute arbitrary code, overwrite files and allow unauthorized disclosure of information. Regardless of the warning, in December of 2010, nearly a year later, approximately 11,000 different organizations downloaded the vulnerable version of Jetty from the Central Repository in a single month. [vi]

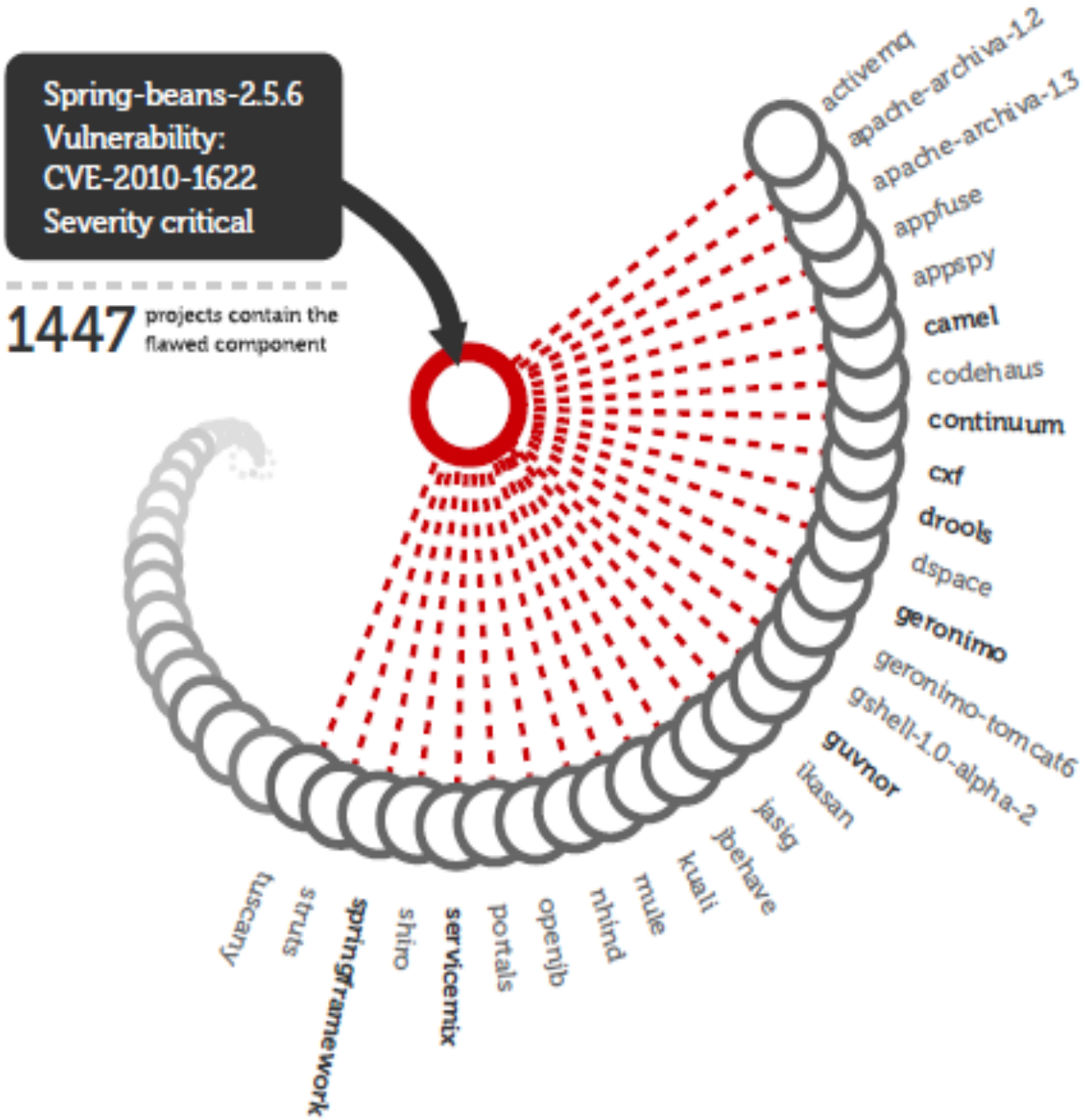
**2 years after a vulnerability was discovered, organizations continue to download the flawed version of Bouncy Castle**

Making the problem harder to deal with is the fact that a single vulnerability in a popular low level component may be used in hundreds, if not thousands of other commonly used open source projects as illustrated in Figure 6. You might not even be aware that your application uses the vulnerable component because it is buried multiple layers down in the open source component stack.




*Source: Maximizing Benefits and Mitigating Risks of Open Source Components in Application Development, by Sonatype*

Even after vulnerabilities are discovered and patches made available, many developers use the flawed, non-patched version of reused components




**Who makes risk decisions?**

**Who inherits the residual risk?**

**Who 'owns' the residual risk attributable to exploitable software?**

*Source: Maximizing Benefits and Mitigating Risks of Open Source Components in Application Development, by Sonatype*

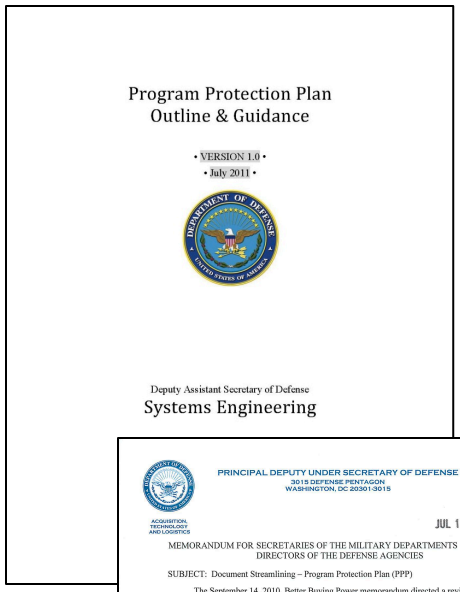
# Challenges in Mitigating Risks Attributable to Exploitable Software and Supply Chains (cont.)

Enterprises seek comprehensive capabilities to:

- Avoid accepting software with **MALWARE** pre-installed. **MAEC**
- Determine that no publicly reported **VULNERABILITIES** remain in code prior to operational acceptance, and that future discoveries of common vulnerabilities and exposures can be quickly patched. **CVE**
- Determine that exploitable software **WEAKNESSES** that put the users most at risk are mitigated prior to operational acceptance or after put into use (and not previously evaluated for exploit potential). **CWE**

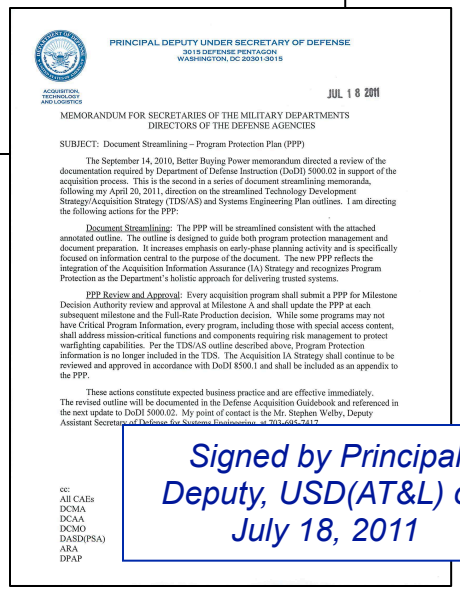
Homeland Security

# Program Protection Plan Outline and Guidance as "Expected Business Practice"


Program Protection Plan
Outline & Guidance

• VERSION 1.0 •
• July 2011 •

Deputy Assistant Secretary of Defense
Systems Engineering


PRINCIPAL DEPUTY UNDER SECRETARY OF DEFENSE
3015 DEFENSE PENTAGON
WASHINGTON, DC 20301-3015

ACQUISITION, TECHNOLOGY AND LOGISTICS

JUL 18 2011

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
DIRECTORS OF THE DEFENSE AGENCIES

SUBJECT: Document Streamlining – Program Protection Plan (PPP)

The September 14, 2010, Better Buying Power memorandum directed a review of the documentation required by Department of Defense Instruction (DoDI) 5000.02 in support of the acquisition process. This is the second in a series of document streamlining memoranda, following my April 20, 2011, direction on the streamlined Technology Development Strategy/Acquisition Strategy (TDS/AS) and Systems Engineering Plan outlines. I am directing the following actions for the PPP:

Document Streamlining: The PPP will be streamlined consistent with the attached annotated outline. The outline is designed to guide both program protection management and document preparation. It increases emphasis on early-phase planning activity and is specifically focused on information central to the purpose of the document. The new PPP reflects the integration of the Acquisition Information Assurance (IA) Strategy and recognizes Program Protection as the Department's holistic approach for delivering trusted systems.

PPP Review and Approval: Every acquisition program shall submit a PPP for Milestone Decision Authority review and approval at Milestone A and shall update the PPP at each subsequent milestone and the Full-Rate Production decision. While some programs may not have Critical Program Information, every program, including those with special access content, shall address mission-critical functions and components requiring risk management to protect warfighting capabilities. Per the TDS/AS outline described above, Program Protection information is no longer included in the TDS. The Acquisition IA Strategy shall continue to be reviewed and approved in accordance with DoDI 8500.1 and shall be included as an appendix to the PPP.

These actions constitute expected business practice and are effective immediately. The revised outline will be documented in the Defense Acquisition Guidebook and referenced in the next update to DoDI 5000.02. My point of contact is the Mr. Stephen Welby, Deputy Assistant Secretary of Defense for Systems Engineering, at 703-695-7417.

cc:
All CAEs
DCMA
DCAA
DCMO
DASD(PSA)
ARA
DPAP

*Signed by Principal Deputy, USD(AT&L) on July 18, 2011*

## What's in the DoD Policy Memo?

– *"Every acquisition program shall submit a PPP for Milestone Decision Authority review and approval at Milestone A and shall update the PPP at each subsequent milestone and the Full-Rate Production decision."*

– Expected business practice, effective immediately, and reflected in upcoming DoDI 5000.02 and DAG updates

The PPP is the Single Focal Point for All Security Activities on the Program

http://www.acq.osd.mil/se/pg/index.html#PPP

# Software Assurance Methods

Counter-measure Selection

## Development Process

Apply assurance activities to the procedures and structure imposed on software development

## Operational System

Implement countermeasures to the design and acquisition of end-item software products and their interfaces

## Development Environment

Apply assurance activities to the environment and tools for developing, testing, and integrating software code and interfaces

**Table 5.3-5-5: Application of Software Assurance Countermeasures (sample)**

| Development Process | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Software (CPI, critical function components, other software)** | **Static Analysis p/a** | **Design Inspect** | **Code Inspect p/a** | **CVE p/a** | **CAPEC p/a** | **CWE p/a** | **Pen Test** | **Test Coverage p/a** |
| Developmental CPI SW | 100/80% | Two Levels | 100/80 | 100/60 | 100/60 | 100/60 | Yes | 75/50% |
| Developmental Critical Function SW | 100/80% | Two Levels | 100/80 | 100/70 | 100/70 | 100/70 | Yes | 75/50% |
| Other Developmental SW | none | One level | 100/65 | 10/0 | 10/0 | 10/0 | No | 50/25% |
| COTS CPI and Critical Function SW | Vendor SwA | Vendor SwA | Vendor SwA | 0 | 0 | 0 | Yes | UNK |
| COTS (other than CPI and Critical Function) and NDI SW | No | No | No | 0 | 0 | 0 | No | UNK |

| Operational System | | | | | | |
|---|---|---|---|---|---|---|
| | **Failover Multiple Supplier Redundancy** | **Fault Isolation** | **Least Privilege** | **System Element Isolation** | **Input checking / validation** | **SW load key** |
| Developmental CPI SW | 30% | All | all | yes | All | All |
| Developmental Critical Function SW | 50% | All | All | yes | All | all |
| Other Developmental SW | none | Partial | none | None | all | all |
| COTS (CPI and CF) and NDI SW | none | Partial | All | None | Wrappers/ all | all |

| Development Environment | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **SW Product** | **Source** | **Release testing** | **Generated code inspection p/a** | | | | | |
| C Compiler | No | Yes | 50/20 | | | | | |
| Runtime libraries | Yes | Yes | 70/none | | | | | |
| Automated test system | No | Yes | 50/none | | | | | |
| Configuration management system | No | Yes | NA | | | | | |
| Database | No | Yes | 50/none | | | | | |
| | | | | | | | | |
| Development Environment Access | Controlled access; Cleared personnel only | | | | | | | |

*Additional Guidance in PPP Outline and Guidance*

# automation can help…


- Common Weakness Enumeration (**CWE**)
- Common Attack Pattern Enumeration and Classification (**CAPEC**)
- CWE Coverage Claims Representation (**CCR**)


- Common Weakness Enumeration (**CWE**)
- Common Weakness Risk Analysis Framework (**CWRAF**)
- Common Weakness Scoring System (**CWSS**)
- Common Attack Pattern Enumeration and Classification (**CAPEC**)
- CWE Coverage Claims Representation (**CCR**)


- Common Vulnerabilities and Exposures (**CVE**)
- Open Vulnerability Assessment Language (**OVAL**)
- Malware Attribute Enumeration and Characterization **(MAEC)**
- Cyber Obersvables eXpression **(CybOX)**

# Software Assurance

Software Assurance (SwA) is **the level of confidence that software functions as intended and is free from vulnerabilities** either intentionally or unintentionally designed or inserted as part of the software throughout the life cycle.*

*Derived From: CNSSI-4009*

## Automation

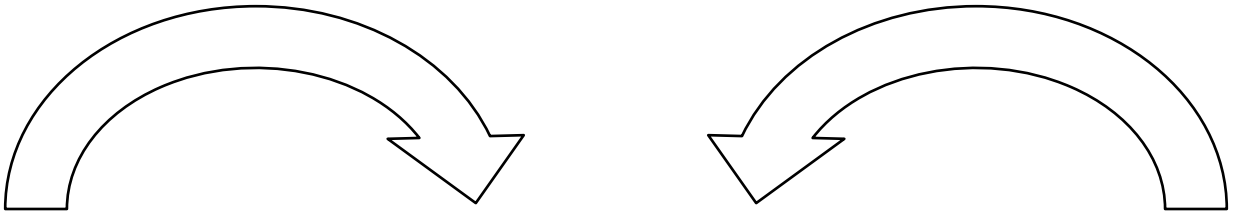
Languages, enumerations, registries, tools, and repositories

## throughout the Lifecycle

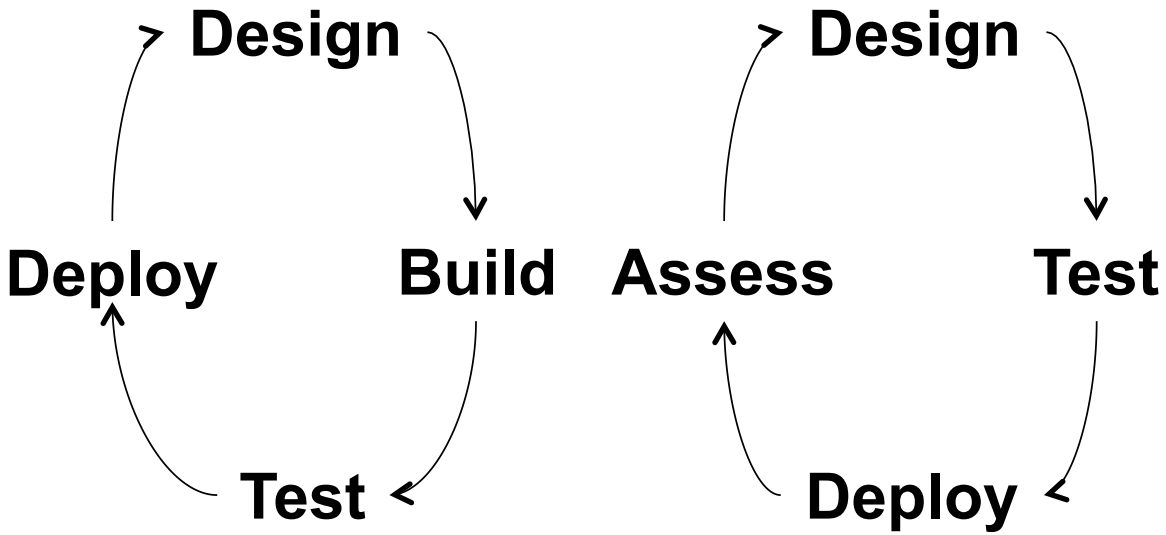
Including design, coding, testing, deployment, configuration and operation

# Making Security Measurable (MSM)

## "You Are Here"



**Software Assurance** | **Enterprise Security Management** | **Threat Management**





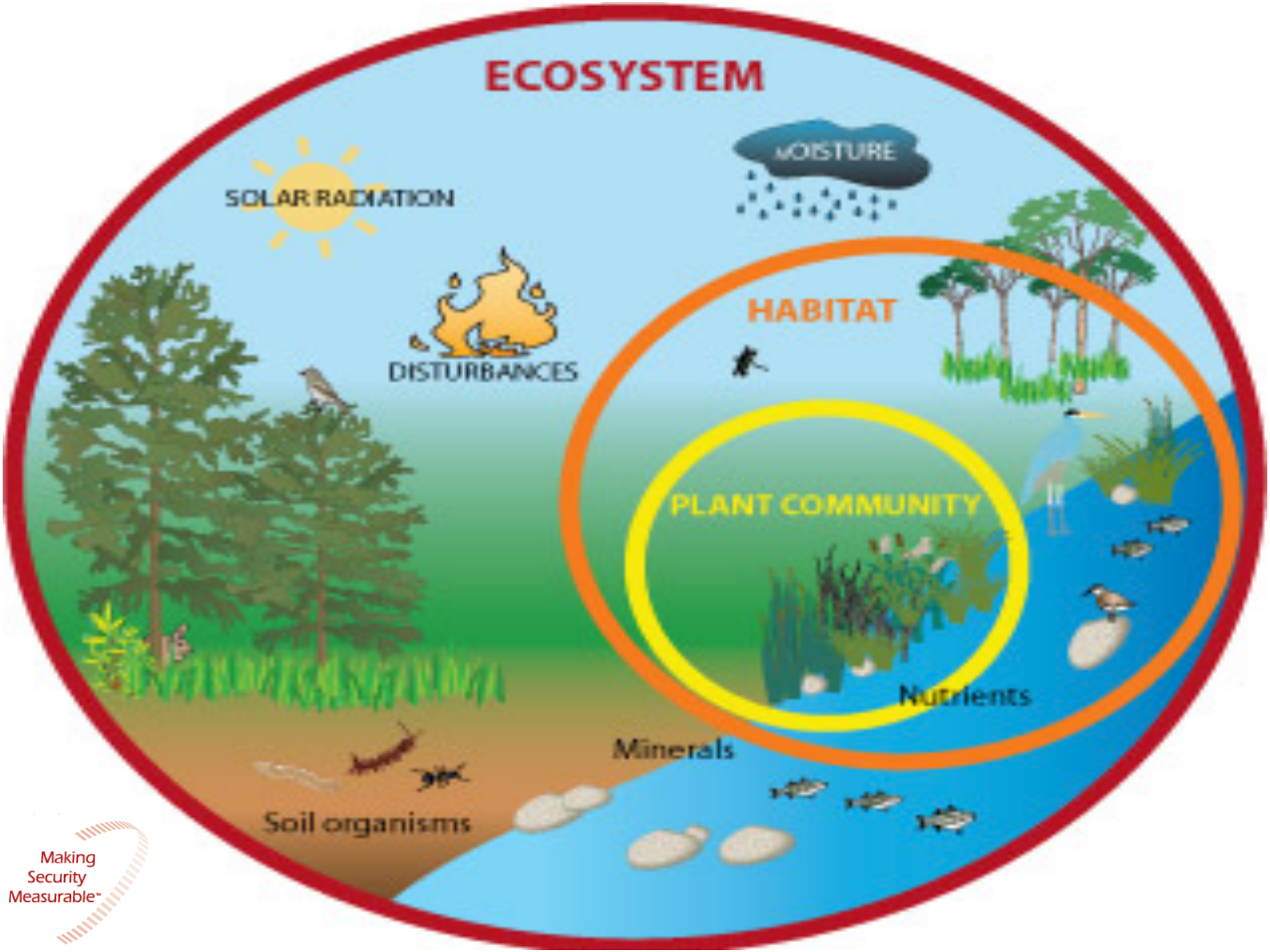


- Vulnerabilities
- Exploits
- Attacks
- Malware

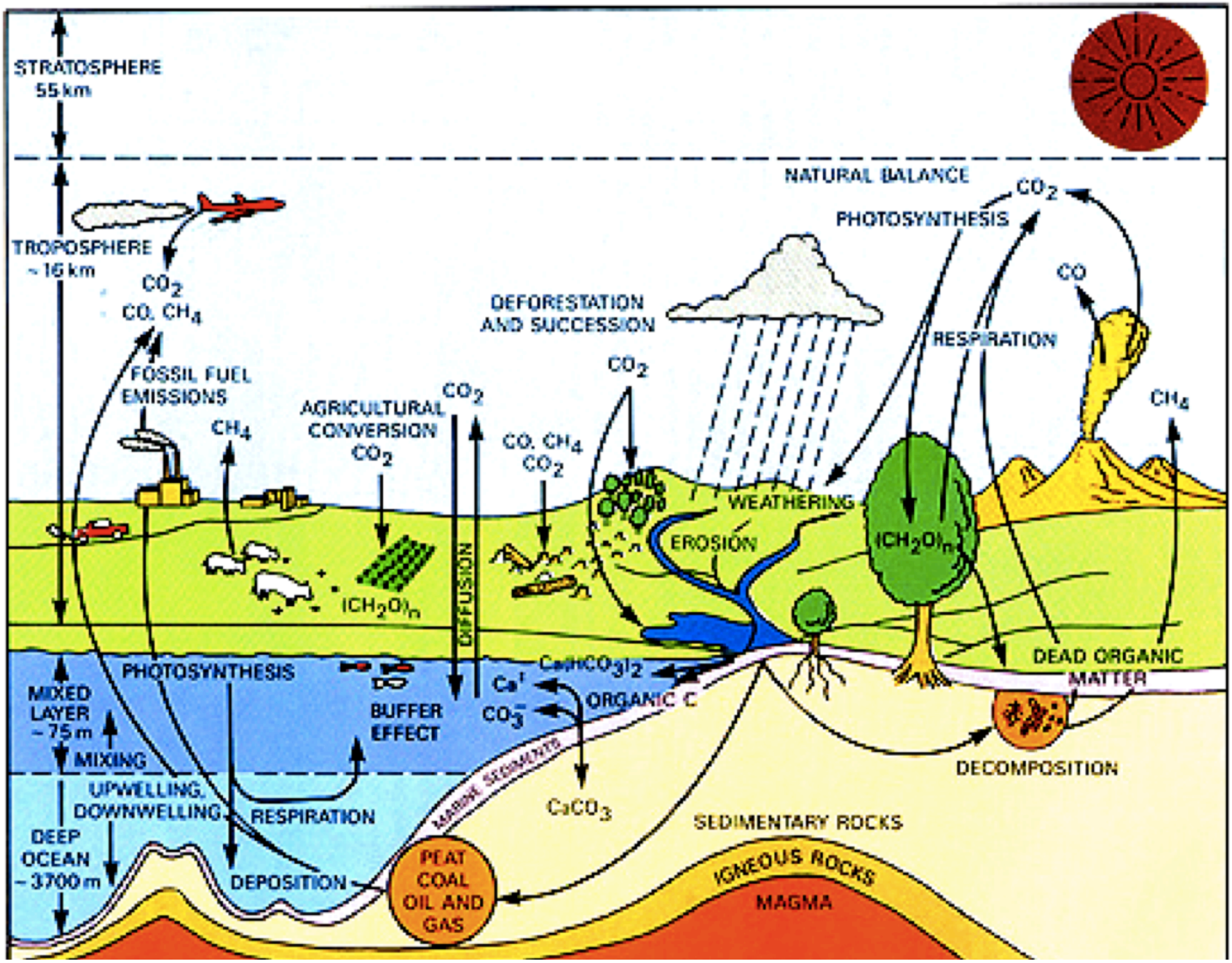
**CWE, CAPEC, CWSS, CWRAF**

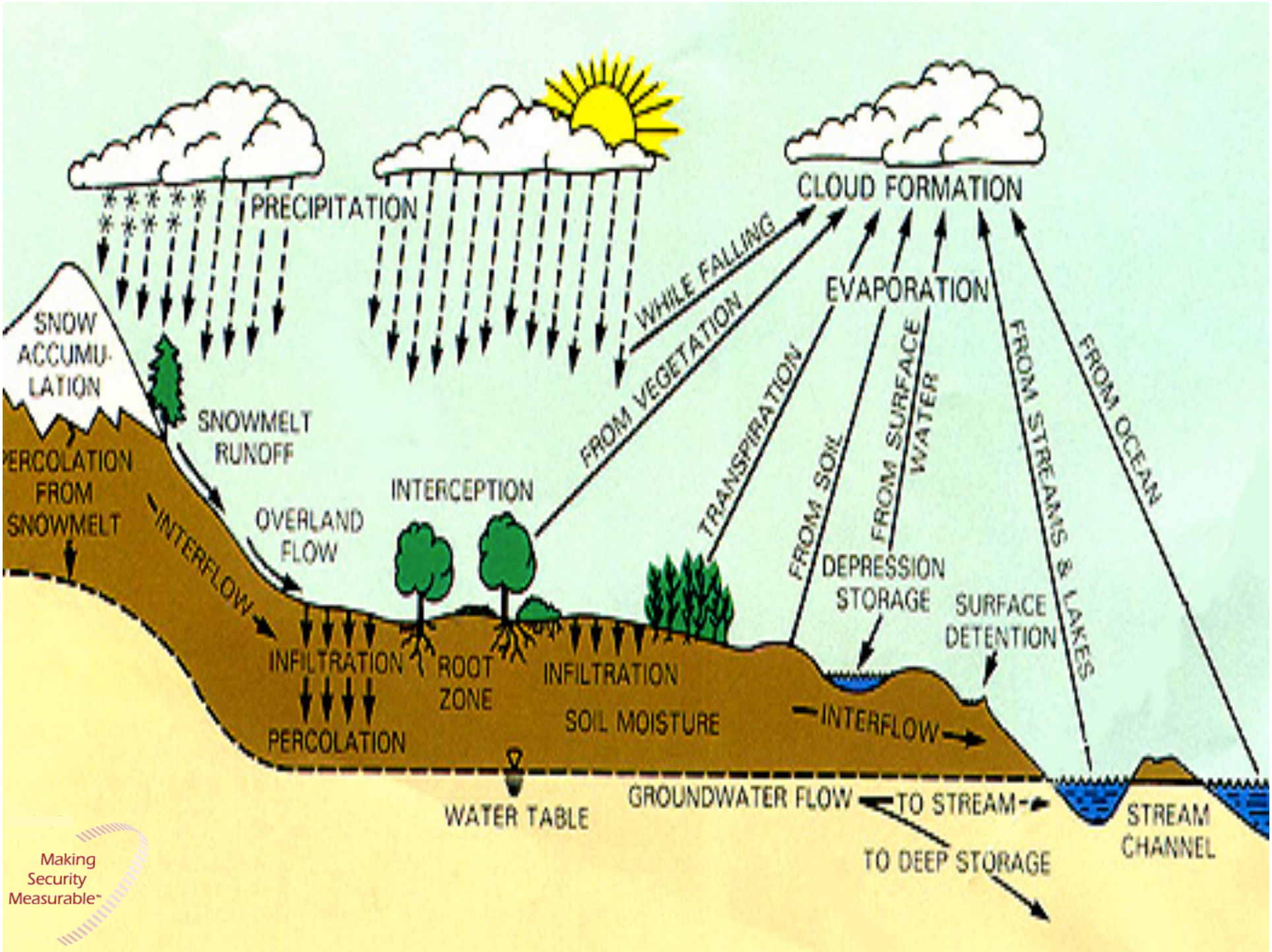
**CPE, CCE, OVAL, OCIL, XCCDF, AssetId, ARF**

**CVE, CWE, CAPEC, MAEC, CybOX, IODEF, RID, RID-T, CYBEX**

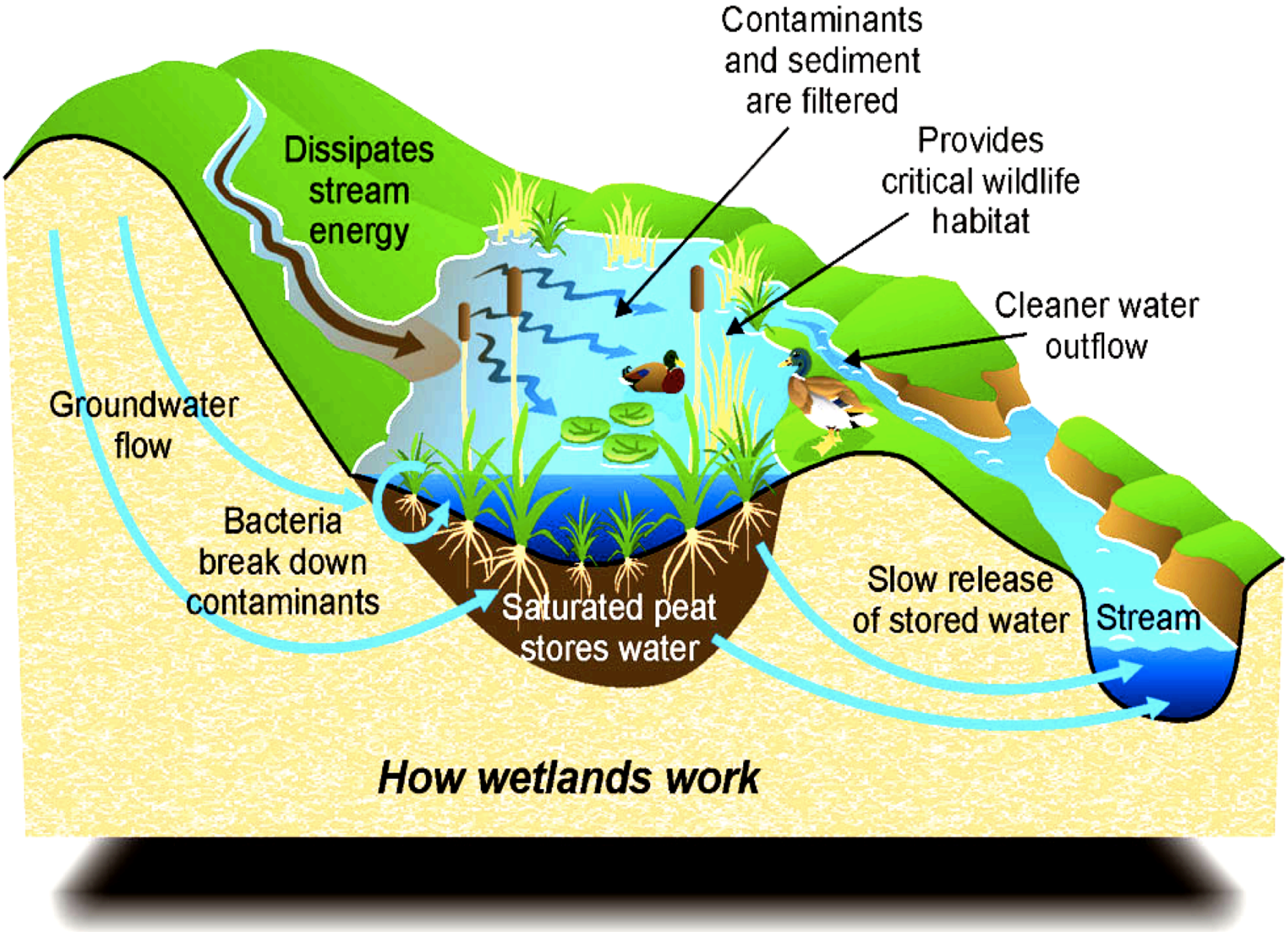


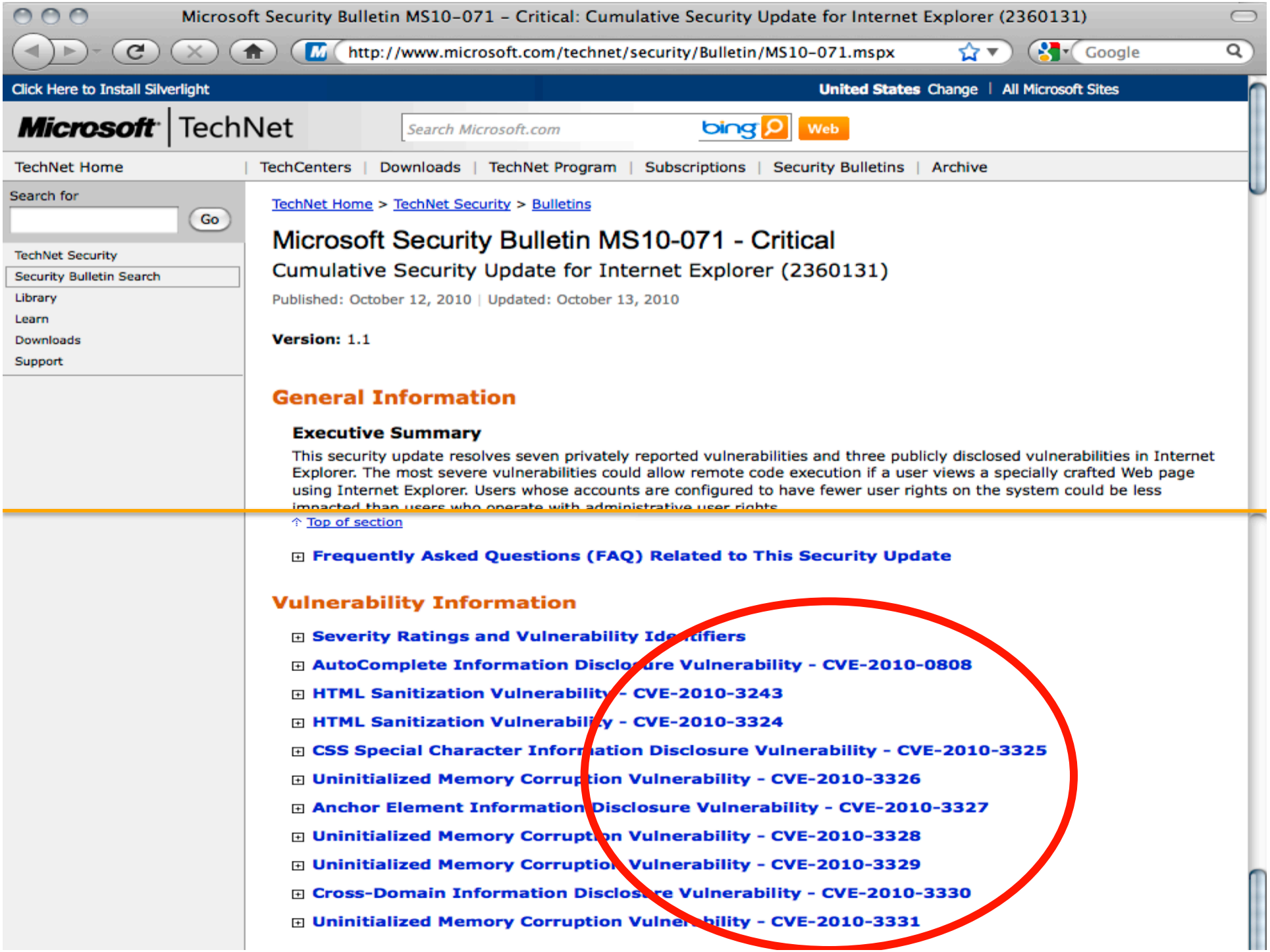

Making Security Measurable™


STRATOSPHERE
55 km
TROPOSPHERE
~ 16 km
$CO_2$
CO, $CH_4$
FOSSIL FUEL
EMISSIONS
$CH_4$
AGRICULTURAL
CONVERSION
$CO_2$
$(CH_2O)_n$
$CO_2$
DIFFUSION
CO, $CH_4$
$CO_2$
DEFORESTATION
AND SUCCESSION
$CO_2$
NATURAL BALANCE
$CO_2$
PHOTOSYNTHESIS
CO
RESPIRATION
$CH_4$
WEATHERING
EROSION
$(CH_2O)_n$
DEAD ORGANIC
MATTER
DECOMPOSITION
MIXED
LAYER
~ 75 m
MIXING
PHOTOSYNTHESIS
BUFFER
EFFECT
$Ca(HCO_3)_2$
$Ca^{++}$
$CO_3^{--}$
ORGANIC C
UPWELLING,
DOWNWELLING
RESPIRATION
DEEP
OCEAN
~ 3700 m
DEPOSITION
MARINE SEDIMENTS
$CaCO_3$
SEDIMENTARY ROCKS
PEAT
COAL
OIL AND
GAS
IGNEOUS ROCKS
MAGMA



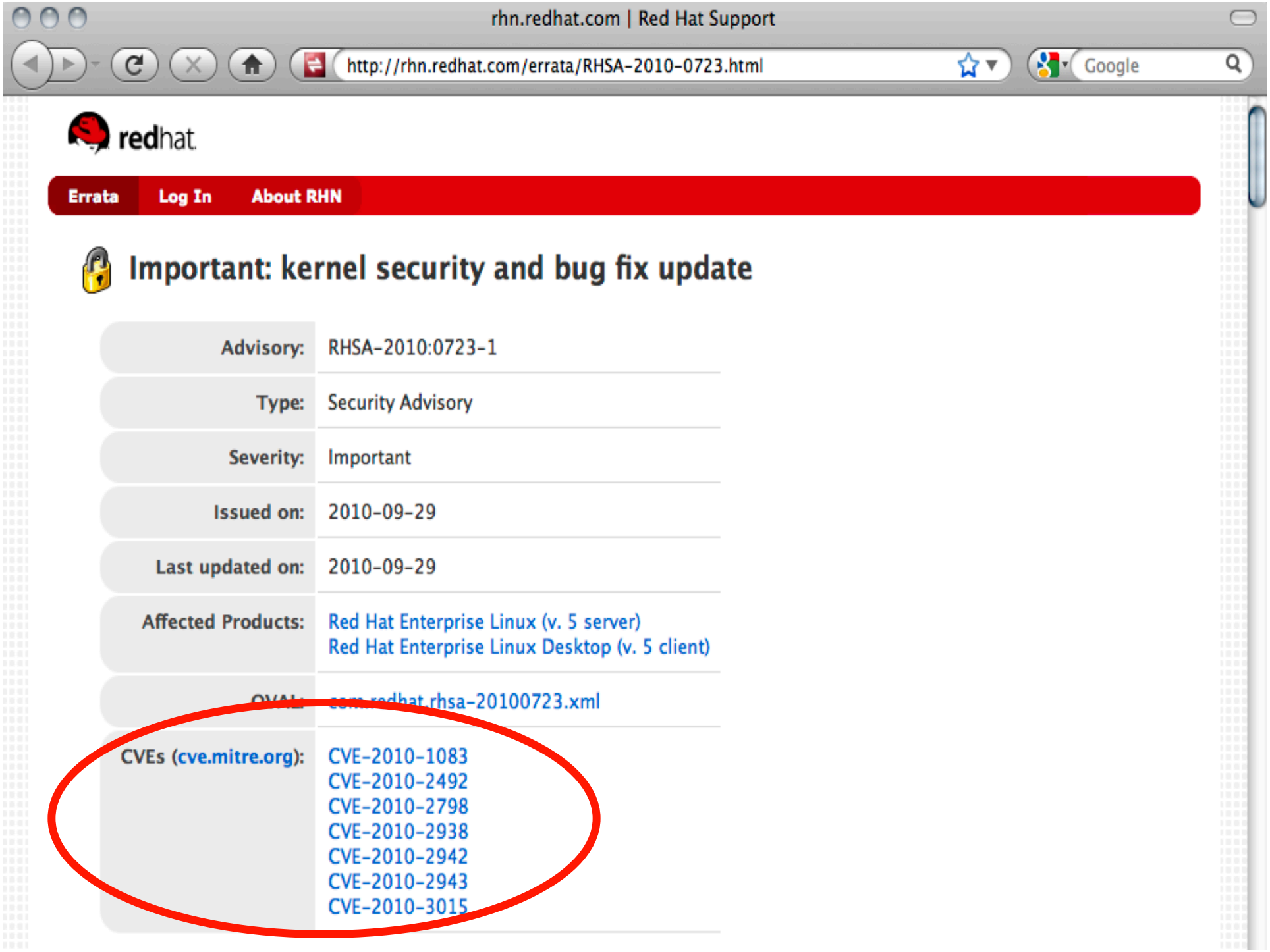

Making Security Measurable™


Contaminants and sediment are filtered
Dissipates stream energy
Provides critical wildlife habitat
Cleaner water outflow
Groundwater flow
Bacteria break down contaminants
Saturated peat stores water
Slow release of stored water
Stream
How wetlands work


Microsoft Security Bulletin MS10-071 – Critical: Cumulative Security Update for Internet Explorer (2360131)
http://www.microsoft.com/technet/security/Bulletin/MS10-071.mspx
Google
Click Here to Install Silverlight
United States Change | All Microsoft Sites
Microsoft TechNet
Search Microsoft.com
bing
Web
TechNet Home | TechCenters | Downloads | TechNet Program | Subscriptions | Security Bulletins | Archive
Search for
Go
TechNet Security
Security Bulletin Search
Library
Learn
Downloads
Support
TechNet Home > TechNet Security > Bulletins
Microsoft Security Bulletin MS10-071 - Critical
Cumulative Security Update for Internet Explorer (2360131)
Published: October 12, 2010 | Updated: October 13, 2010
Version: 1.1
General Information
Executive Summary
This security update resolves seven privately reported vulnerabilities and three publicly disclosed vulnerabilities in Internet Explorer. The most severe vulnerabilities could allow remote code execution if a user views a specially crafted Web page using Internet Explorer. Users whose accounts are configured to have fewer user rights on the system could be less
Top of section
Frequently Asked Questions (FAQ) Related to This Security Update
Vulnerability Information
Severity Ratings and Vulnerability Identifiers
AutoComplete Information Disclosure Vulnerability - CVE-2010-0808
HTML Sanitization Vulnerability - CVE-2010-3243
HTML Sanitization Vulnerability - CVE-2010-3324
CSS Special Character Information Disclosure Vulnerability - CVE-2010-3325
Uninitialized Memory Corruption Vulnerability - CVE-2010-3326
Anchor Element Information Disclosure Vulnerability - CVE-2010-3327
Uninitialized Memory Corruption Vulnerability - CVE-2010-3328
Uninitialized Memory Corruption Vulnerability - CVE-2010-3329
Cross-Domain Information Disclosure Vulnerability - CVE-2010-3330
Uninitialized Memory Corruption Vulnerability - CVE-2010-3331


**Oracle Database Server Risk Matrix**

| CVE# | Component | Protocol | Package and/or Privilege Required | Remote Exploit without Auth.? | CVSS VERSION 2.0 RISK (see Risk Matrix Definitions) | | | | | | | Last Affected Patch set (per Supported Release) | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Base Score | Access Vector | Access Complexity | Authen-tication | Confiden-tiality | Integrity | Avail-ability | | |
| *CVE-2010-2390 (Oracle Enterprise Manager Grid Control)* | EM Console | HTTP | None | Yes | 7.5 | Network | Low | None | Partial+ | Partial+ | Partial+ | 10.1.0.5, 10.2.0.3 | See Note 1 |
| CVE-2010-2419 | Java Virtual Machine | Oracle Net | Create Session | No | 6.5 | Network | Low | Single | Partial+ | Partial+ | Partial+ | 10.1.0.5, 10.2.0.4, 11.1.0.7, 11.2.0.1 | |
| CVE-2010-1321 | Change Data Capture | Oracle Net | Execute on DBMS_CDC_PUBLISH | No | 5.5 | Network | Low | Single | Partial+ | Partial+ | None | - | See Note 2 |
| CVE-2010-2412 | OLAP | Oracle Net | Create Session | No | 5.5 | Network | Low | Single | Partial+ | Partial+ | None | 11.1.0.7 | |
| CVE-2010-2415 | Change Data Capture | Oracle Net | Execute on DBMS_CDC_PUBLISH | No | 4.9 | Network | Medium | Single | Partial+ | Partial+ | None | 10.1.0.5, 10.2.0.4, 11.1.0.7, 11.2.0.1 | |
| CVE-2010-2411 | Job Queue | Oracle Net | Execute on SYS.DBMS_IJOB | No | 4.6 | Network | High | Single | Partial+ | Partial+ | Partial+ | - | See Note 2 |
| CVE-2010-2407 | JDK | HTTP | None | Yes | 4.3 | Network | Medium | None | None | Partial | None | 10.1.0.5, 10.2.0.4, 11.1.0.7 | |
| CVE-2010-2391 | Core RDBMS | Oracle Net | Create Session | No | 3.6 | Network | High | Single | Partial | Partial | None | 10.1.0.5, 10.2.0.3 | |
| *CVE-2010-2389 (Oracle Fusion Middleware)* | Perl | Oracle Net | Local Logon | No | 1.0 | Local | High | Single | None | Partial+ | None | - | See Note 2 |


rhn.redhat.com | Red Hat Support
http://rhn.redhat.com/errata/RHSA-2010-0723.html
Google
redhat
Errata
Log In
About RHN
Important: kernel security and bug fix update
Advisory: RHSA-2010:0723-1
Type: Security Advisory
Severity: Important
Issued on: 2010-09-29
Last updated on: 2010-09-29
Affected Products: Red Hat Enterprise Linux (v. 5 server)
Red Hat Enterprise Linux Desktop (v. 5 client)
CVEs (cve.mitre.org): CVE-2010-1083
CVE-2010-2492
CVE-2010-2798
CVE-2010-2938
CVE-2010-2942
CVE-2010-2943
CVE-2010-3015

# APPLE-SA-2010-08-11-1 iOS 4.0.2 Update for iPhone and iPod touch

**Subject**: APPLE-SA-2010-08-11-1 iOS 4.0.2 Update for iPhone and iPod touch
From: Apple Product Security <email@hidden>
Date: Wed, 11 Aug 2010 12:19:43 -0700
Delivered-to: email@hidden
Delivered-to: email@hidden

---

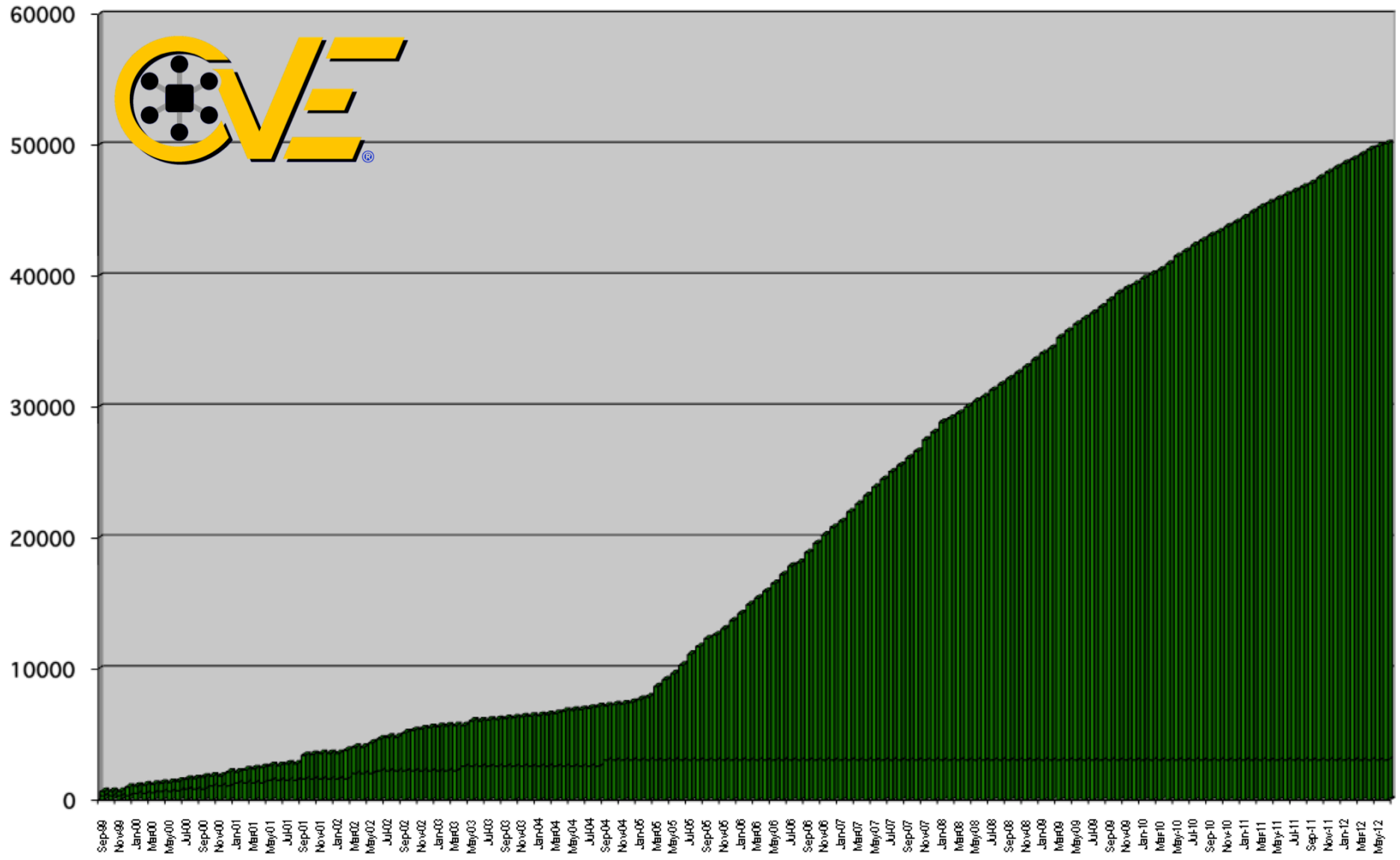
```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

APPLE-SA-2010-08-11-1 iOS 4.0.2 Update for iPhone and iPod touch

iOS 4.0.2 Update for iPhone and iPod touch is now available and
addresses the following:

FreeType
CVE-ID:  CVE-2010-1797
Available for:  iOS 2.0 through 4.0.1 for iPhone 3G and later,
iOS 2.1 through 4.0 for iPod touch (2nd generation) and later
Impact:  Viewing a PDF document with maliciously crafted embedded
fonts may allow arbitrary code execution
Description:  A stack buffer overflow exists in FreeType's handling
of CFF opcodes. Viewing a PDF document with maliciously crafted
```

# CVE 1999 to 2012



## Advisory Issuers Using CVE Identifiers

Trustwave · IBM · EMC² · iDEFENSE A VeriSign Company · Novell · 3com · RAPID7 · NISCC · Secunia Stay Secure · ca · Carnegie Mellon Software Engineering Institute · debian · SuSE · Cisco Systems · Foundstone · redhat · Adobe · gentoo linux · symantec. · Microsoft · Juniper Networks · FreeBSD · CORE Security Technologies · ORACLE · Sun microsystems · hp · MIT Lincoln Laboratory · Google

# US-CERT

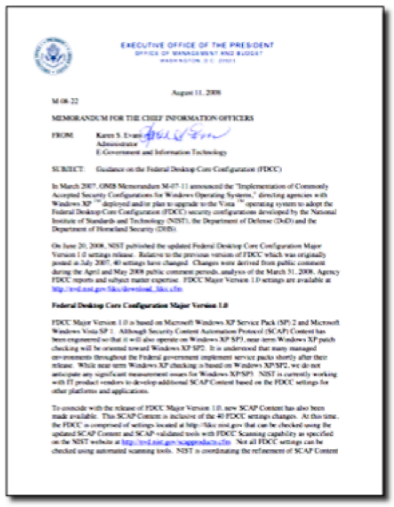
**UNITED STATES COMPUTER EMERGENCY READINESS TEAM**

Cyber Security Bulletin - Technical Cyber Security Alerts - Cyber Security Alerts

CVE COMPATIBLE

**286 Products and Services from 157 Organizations in 26 Countries**

**Information Assurance Vulnerability Management (IAVM)**



EXECUTIVE OFFICE OF THE PRESIDENT

August 11, 2008

M-08-22

MEMORANDUM FOR THE CHIEF INFORMATION OFFICERS

FROM: Karen S. Evans
Administrator
E-Government and Information Technology

SUBJECT: Guidance on the Federal Desktop Core Configuration (FDCC)

In March 2007, OMB Memorandum M-07-11 announced the "Implementation of Commonly Accepted Security Configurations for Windows Operating Systems," directing agencies with Windows XP ™ deployed and/or plan to upgrade to the Vista ™ operating system to adopt the Federal Desktop Core Configuration (FDCC) security configurations developed by the National Institute of Standards and Technology (NIST), the Department of Defense (DoD) and the Department of Homeland Security (DHS).

On June 20, 2008, NIST published the updated Federal Desktop Core Configuration Major Version 1.0 settings release. Relative to the previous version of FDCC which was originally posted in July 2007, 40 settings have changed. Changes were derived from public comment during the April and May 2008 public comment periods, analyses of the March 31, 2008, Agency FDCC reports and subject matter expertise. FDCC Major Version 1.0 settings are available at http://nvd.nist.gov/fdcc/download_fdcc.cfm.

**Federal Desktop Core Configuration Major Version 1.0**

FDCC Major Version 1.0 is based on Microsoft Windows XP Service Pack (SP) 2 and Microsoft Windows Vista SP 1. Although Security Content Automation Protocol (SCAP) Content has been engineered so that it will also operate on Windows XP SP3, near-term Windows XP patch checking will be oriented toward Windows XP SP2. It is understood that many managed environments throughout the Federal government implement service packs shortly after their release. While near-term Windows XP checking is based on Windows XP SP2, we do not anticipate any significant measurement issues for Windows XP SP3. NIST is currently working with IT product vendors to develop additional SCAP Content based on the FDCC settings for other platforms and applications.

To coincide with the release of FDCC Major Version 1.0, new SCAP Content has also been made available. This SCAP Content is inclusive of the 40 FDCC settings changes. At this time, the FDCC is comprised of settings located at http://fdcc.nist.gov that can be checked using the updated SCAP Content and SCAP-validated tools with FDCC Scanning capability as specified on the NIST website at http://nvd.nist.gov/scapproducts.cfm. Not all FDCC settings can be checked using automated scanning tools. NIST is coordinating the refinement of SCAP Content

SCAP NIST

NIST Security Configuration CHECKLISTS http://checklists.nist.gov CSO

OVAL

**ADOPTER**

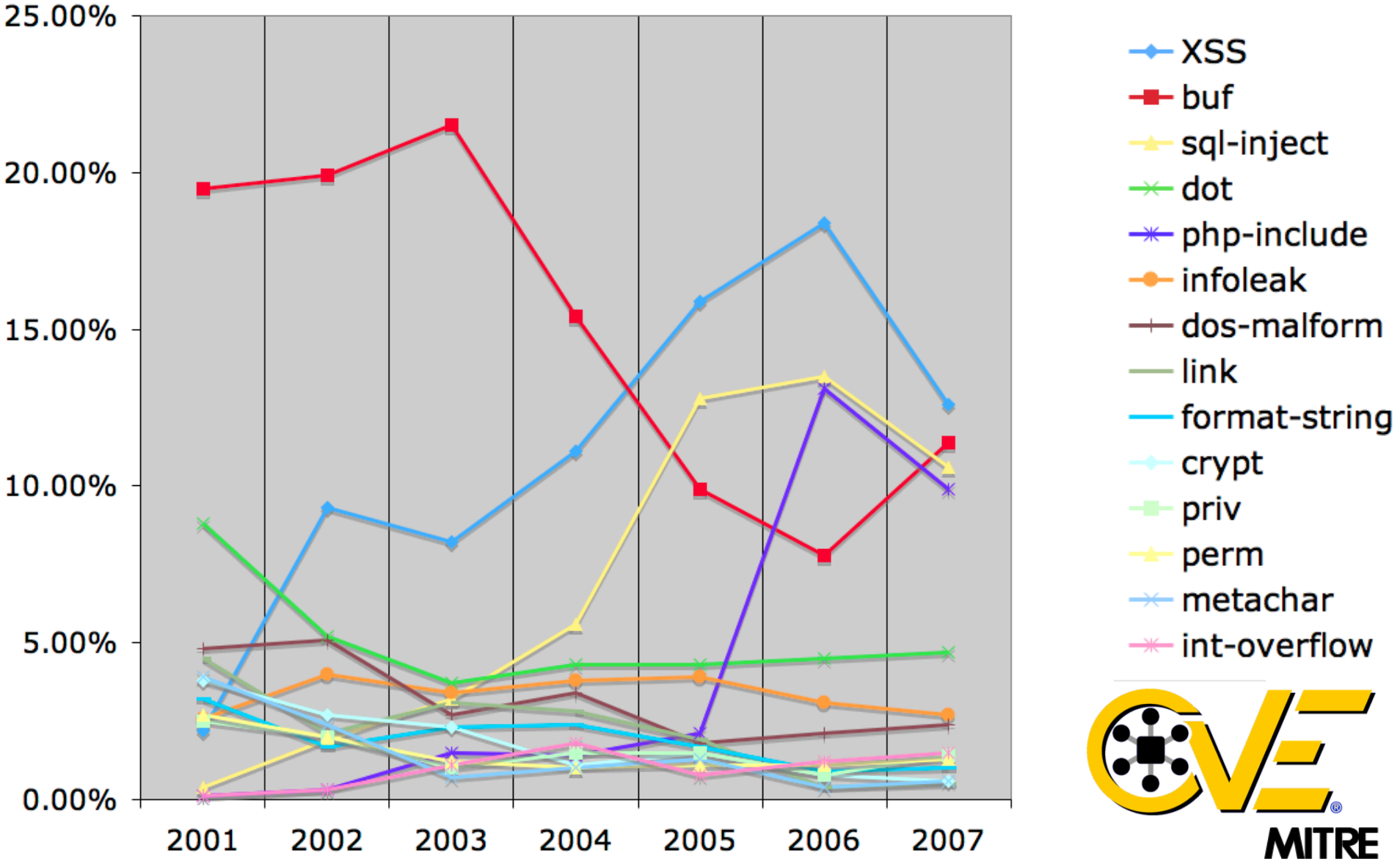
**45 Products and Services from 31 Organizations in 7 Countries**

**Assured Compliance Assessment Solution (ACAS) Host Based Security Solution (HBSS)**

Public Repositories of OVAL content

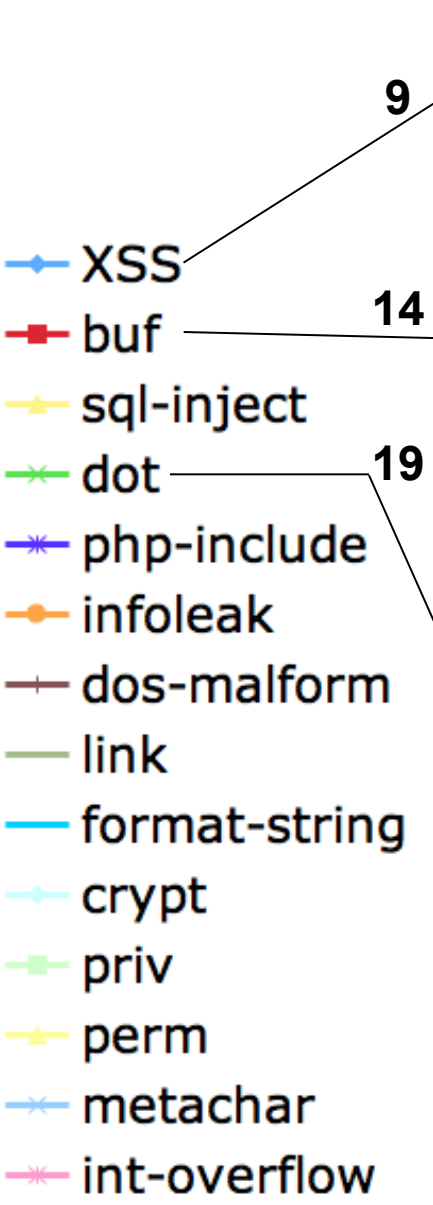
debian · NIST · Novell. · redhat · secpod Security Simplified

# Vulnerability Type Trends:
# A Look at the CVE List (2001 - 2007)




CVE®
MITRE

# Removing and Preventing the Vulnerabilities Requires More Specific Definitions…CWEs




9

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)

14

Improper Restriction of Operations within the Bounds of a Memory Buffer (119)
- Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (120)
- Write-what-where Condition (123)
- 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)

19

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)


Weakness
CWE
Research
Concepts
primary
Content
Description
ID
Name
MITRE
Team
updated
Development
Relationships
ParentOf
Taxonomy
Internal
Improper
Base
Source
Variant
ChildOf
Notes
Example
ClassWeakness
Mapped
Implementation
Weaknesses
Organization
Date
Summary
History
Modifier
Submitter
Concepts1000
CategoryCategory
BaseWeakness
Node
Mappings
View
Mitigations
Potential
Code
Platforms
application
weakness
Demonstrative
External
Class
input
Time
using
Architecture
Externally
Java
Languages
Mined
Change
Definition
validation
Addressed
Standard
Previous
Modifications
Injection
Secure
Top
code
Window
data
Examples
New
Phase
used
Dalci
Type
Common
Data
request
server
memory
VariantWeakness
Attack
system
Submission
inputs
Coding
Nature
use
Modification
file
pertains
user
toView
relationship
Entry
may
files
Failure
buffer
web
Fit
Control
Observed
Draft
Bad
error
software
Design
program
Neutralization
References
Input
Category
following
allows
Extended
might
value
Related
function
malicious
OWASP
control
xss
example
Patterns
attacker
Applicable
weaknesses
Status
Use
Submissions
Language
PLOVER
public
access
Consequences
information


Wouldn’t it be nice
if the weaknesses
in software were as
easy to spot and
their impact as
easy to understand
as a screen door in
a submarine…


a time-of-check-time-of-use (TOCTOU) bug that led to code calling into a freed memory block. The
on Weakness Enumeration (CWE) classification for this vulnerability is CWE-367.

Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

CWE and SANS Institute TOP 25 MOST DANGEROUS SOFTWARE ERRORS



CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
Community
Related Activities
Discussion List
Research
CWE/SANS Top 25
CWSS
News
Calendar
Free Newsletter
Compatibility
Program
Requirements
Declarations
Make a Declaration
Contact Us
Search the Site

# CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

## Time-of-check Time-of-use (TOCTOU) Race Condition

**Weakness ID:** 367 *(Weakness Base)* **Status:** Incomplete

### Description

**Description Summary**

The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.

**Extended Description**

This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multithreaded programs.

### Alternate Terms

**TOCTTOU:** The TOCCTOU acronym expands to "Time Of Check To Time Of Use". Usage varies between TOCTOU and TOCTTOU.

### Time of Introduction

- Implementation

### Applicable Platforms

**Languages**

All

### Common Consequences

| Scope | Effect |
|---|---|
| Access Control | The attacker can gain access to otherwise unauthorized resources. |
| Access Control Authorization | Race conditions such as this kind may be employed to gain read or write access to resources which are not normally readable or writable by the user in question. |
| Integrity | The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user. |
| Accountability | If a file or other resource is written in this method, as opposed to in a valid way, logging of the activity may not occur. |
| Non-Repudiation | In some cases it may be possible to delete files a malicious user might not otherwise have access to, such as log files. |

**IBM Software**
Technical White Paper

One way to improve software security is to gain a better understanding of the most common weaknesses that can affect software security. With that in mind, there are many resources available online to help organizations learn about

**Resources available to help organizations protect systems in**

| Resource | Focus |
|---|---|
| DoD Information Assurance Certification and Accreditation Process (DIACAP) | The DIACAP defines the minimum standa accredited by the DoD and authorized to application-level security controls, but it i activities, general tasks, and a managem |
| Defense Information Systems Agency (DISA) | The DISA provides a security technical in development that offer more granular inf bility assessment techniques. The checklist is the same one used by DoD auditors. |
| U.S. Department of Homeland Security (DHS) | The DHS offers information on security best practices and tools for application- and soft part of its "Build Security In" initiative. |
| The Common Weaknesses Enumeration project, a community-based program sponsored by the MITRE Corporation, an IBM Business Partner | The MITRE Corporation maintains the online common vulnerabilities and exposures (CVE enumeration (CWE) knowledge bases about currently known vulnerabilities and types of knowledge base focuses on packaged software and deals with patches and known vuln knowledge base focuses on code vulnerabilities. |
| The Open Web Application Security Project (OWASP) | One of the best sources for information on web application security issues, the OWASP 10 list of the most dangerous and most commonly found and commonly exploited vulne how to identify, fix and avoid them. |
| Cigital Building Security In Maturity Model (BSIMM) | Created by Cigital, an IBM Business Partner, the BSIMM is designed to help organizatio and plan a software security initiative. The focus is on making applications more secure, process and at later stages in the software life cycle. |
| IBM X-Force™ research and development team | A global cyberthreat and risk analysis team that monitors traffic and attacks around the IBM X-Force team is an excellent resource for trend analysis and answers to questions a attacks are most common, where they are coming from and what organizations can do the risks. |
| IBM Institute for Advanced Security (IAS) | This companywide cybersecurity initiative applies IBM research, services, software and t help governments and other clients improve the security and resiliency of their IT and bu |

## Test and vulnerability assessment

Testing applications for security defects should be an integral and organic part of any software testing process. During security testing, organizations should test to help ensure that the security requirements have been implemented and the product is free of vulnerabilities.

The SEF refers to the MITRE Common Weakness Enumeration[5] (CWE) list and the Common Vulnerability E
be tested. Thi
information ar
and vulnerabi
against the m

Creating a se
plan includes

[5] For more inform
[6] For more inform



Software Engineering Institute

# Making the Business Case for Software Assurance

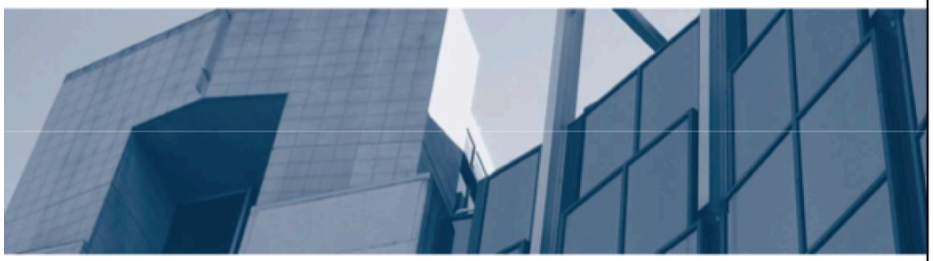
Nancy R. Mead
Julia H. Allen
W. Arthur Conklin
Antonio Drommi
John Harrison
Jeff Ingalsbe
James Rainey
Dan Shoemaker

**April 2009**

**SPECIAL REPORT**
CMU/SEI-2009-SR-001

**CERT Program**


http://www.sei.cmu.edu



Carnegie Mellon

MITRE

# OVM: An Ontology for Vulnerability Management

Ju An Wang & Minzhe Guo
Southern Polytechnic State University
1100 South Marietta Parkway
Marietta, GA 30060
(01) 678-915-3718

jwang@spsu.edu

## ABSTRACT

In order to reach the goals of the Information Security Automation Program (ISAP) [1], we propose an ontological approach to capturing and utilizing the fundamental concepts in information security and their relationship, retrieving vulnerability data and reasoning about the cause and impact of vulnerabilities. Our ontology for vulnerability management (OVM) has been populated with all vulnerabilities in NVD [2] with additional inference rules, knowledge representation, and data-mining mechanisms. With the seamless integration of common vulnerabilities and their related concepts such as attacks and countermeasures, OVM provides a promising pathway to making ISAP successful.

## Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General [Security and protection]; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection;

## General Terms

Ontology, Security, Vulnerability Analysis and Management

## Keywords

Security vulnerability, Semantic technology, Ontology, Vulnerability analysis

## 1. INTRODUCTION

The Information Security Automation Program (ISAP) is a U.S. government multi-agency initiative to enable automation and standardization of technical security operations [1]. Its high-level goals include standards based automation of security checking and remediation as well as automation of technical compliance activities. Its low-level objectives include enabling standards based communication of vulnerability data, customizing and managing configuration baselines for various IT products, assessing information systems and reporting compliance status, using standard metrics to weight and aggregate potential vulnerability impact, and remediating identified vulnerabilities [1]. Secure computer systems ensure that confidentiality, integrity, and availability are maintained for users, data, and other information assets. Over the past a few decades, a significantly large amount of knowledge has been accumulated in the area of information security. However, a lot of concepts in information security are vaguely defined and sometimes they have different semantics in different contexts, causing misunderstanding among stake holders due to the language ambiguity. On the other hand, the standardization, design and development of security tools [1-5] require a systematic classification and definition of security concepts and techniques. It is important to have a clearly defined vocabulary and standardized language as means to accurately communicate system vulnerability information and their countermeasures among all the people involved. We believe that semantic technology in general, and ontology in particular, could be a useful tool for system security. Our research work has confirmed this belief and this paper will report some of our work in this area.

An ontology is a specification of concepts and their relationship. Ontology represents knowledge in a formal and structured form. Therefore, ontology provides a better tool for communication, reusability, and organization of knowledge. Ontology is a knowledge representation (KR) system based on Description Logics (DLs) [6], which is an umbrella name for a family of KR formalisms representing knowledge in various domains. The DL formalism specifies a knowledge domain as the "world" by first defining the relevant concepts of the domain, and then it uses these concepts to specify properties of objects and individuals occurring in the domain [10-12]. Semantic technologies not only provide a tool for communication, but also a foundation for high-level reasoning and decision-making. Ontology, in particular, provides the potential of formal logic inference based on well-defined data and knowledge bases. Ontology captures the relationships between collected data and use the explicit knowledge of concepts and relationships to deduce the implicit and inherent knowledge. As a matter of fact, a heavy-weight ontology could be defined as a formal logic system, as it includes facts and rules, concepts, concept taxonomies, relationships, properties, axioms and constraints.

A vulnerability is a security flaw, which arises from computer system design, implementation, maintenance, and operation. Research in the area of vulnerability analysis focuses on discovery of previously unknown vulnerabilities and quantification of the security of systems according to some metrics. Researchers at MITRE have provided a standard format for naming a security vulnerability, called Common Vulnerabilities and Exposures (CVE) [14], which assigns each vulnerability a unique identification number. We have designed a vulnerability ontology OVM (ontology for vulnerability management) populated with all existing vulnerabilities in NVD [2]. It supports research on reasoning about vulnerabilities and characterization of vulnerabilities and their impact on computing systems. Vendors and users can use our ontology in support of vulnerability analysis, tool development and vulnerability management.

The rest of this paper is organized as follows: Section 2 presents the architecture of our OVM. Section 3 discusses how to populate the OVM with vulnerability instances from NVD and other



CSIIRW '09, April 13-15, Oak Ridge, Tennessee, USA

# 16 July 2010

## A Human Capital Crisis in Cybersecurity

### Technical Proficiency Matters

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

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

based on a body of knowledge that represents the complete set of concepts, terms and activities that make up a professional domain. And absent such a body of knowledge there is little basis for supporting a certification program. Indeed it would be dangerous and misleading.

A complete body of knowledge covering the entire field of software engineering may be years away. However, the body of knowledge needed by professionals to create software free of common and critical security flaws has been developed, vetted widely and kept up to date. That is the foundation for a certification program in software assurance that can gain wide adoption. It was created in late 2008 by a consortium of national experts, sponsored by DHS and NSA, and was updated in late 2009. It contains ranked lists of the most common errors, explanations of why the errors are dangerous, examples of those errors in multiple languages, and ways of eliminating those errors. It can be found at http://cwe.mitre.org/top25.

Any programmer who writes code without being aware of those problems and is not capable of writing code free of those errors is a threat to his or her employers and to others who use computers connected to systems running his or her software.

A complete body of knowledge covering the entire field of software engineering may be years away. However, the body of knowledge needed by professionals to create software free of common and critical security flaws has been developed, vetted widely and kept up to date. That is the foundation for a certification program in software assurance that can gain wide adoption. It was created in late 2008 by a consortium of national experts, sponsored by DHS and NSA, and was updated in late 2009. It contains ranked lists of the most common errors, explanations of why the errors are dangerous, examples of those errors in multiple languages, and ways of eliminating those errors. It can be found at http://cwe.mitre.org/top25.

Any programmer who writes code without being aware of those problems and is not capable of writing code free of those errors is a threat to his or her employers and to others who use computers connected to systems running his or her software.

CSSLP®

The **Certified Secure Software Lifecycle Professional** (CSSLP) Certification Program will show software lifecycle stakeholders not only how to implement security, but how to glean security requirements, design, architect, test and deploy secure software.

## An Overview of the Steps:

**(ISC)² ® 5-day CSSLP CBK® Education Program**

Educate yourself and learn security best practices and industry standards for the software lifecycle through the CSSLP Education Program.(ISC)² provides education your way to fit your life and schedule.Completing this course will, not only teach all of the stablish a security plan across your


COMPUTER BASED TESTING

NOW AVAILABLE FOR THE CSSLP

# Industry Uptake

## Foreword

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

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

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

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

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

### What's New

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

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

SAFECode Driving Security and Integrity

ii

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

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

xample.

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

SAFECode Driving Security and Integrity

Fundamental Practices for Secure Software Development

2ND EDITION

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

February 8, 2011

EDITOR Stacy Simpson, SAFECode

AUTHORS

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

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

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

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

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

CWE References

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

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

CWE

SAFECode Driving Security and Integrity

5

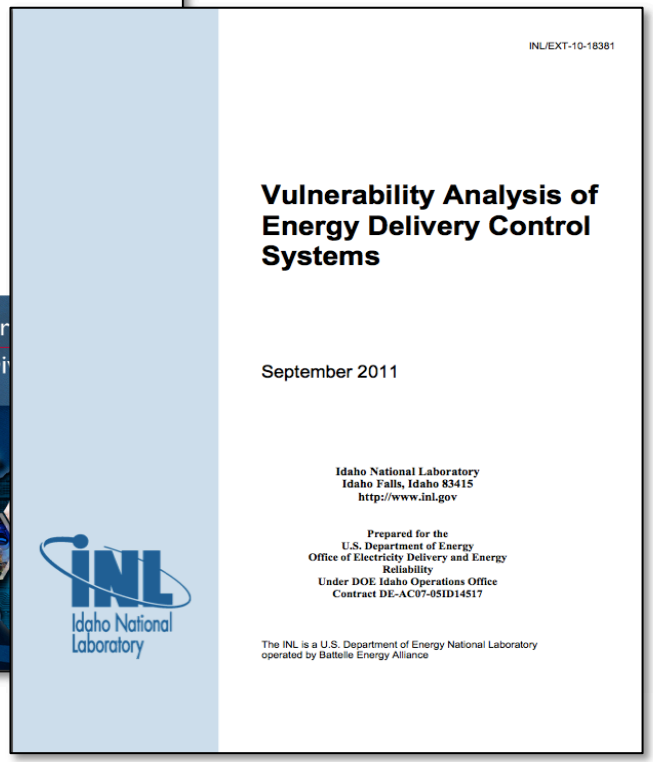
MITRE

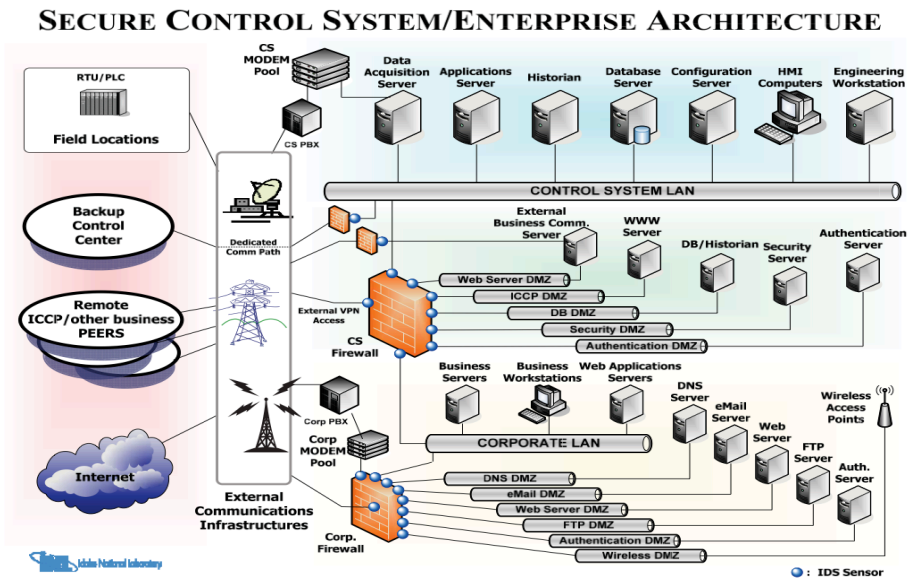
# Control Systems Vulnerabilities – DOE & DHS

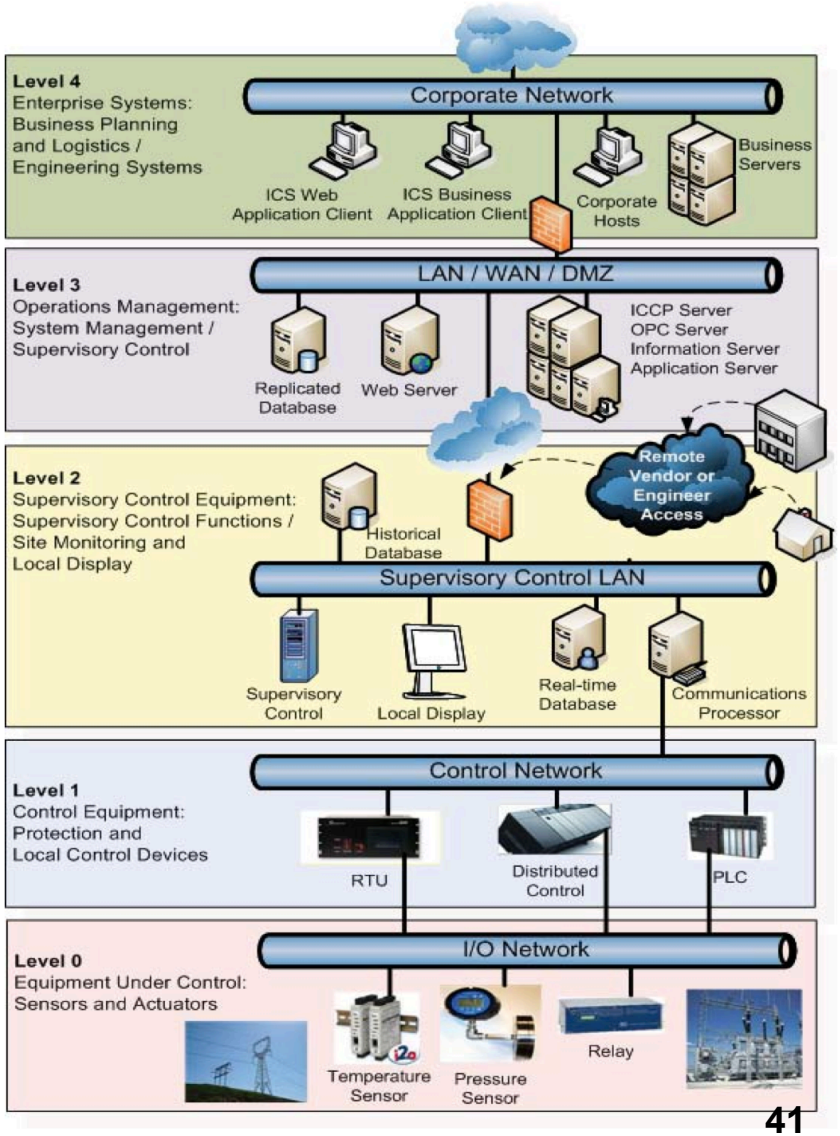
Common Cybersecurity Vulnerabilities in Industrial Control Systems

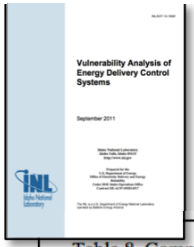
*May 2011*

Homeland Security

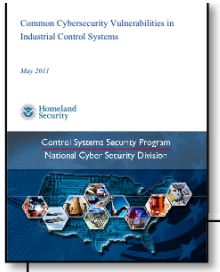



# DOE/INL Common Vulnerabilities…

Table 8. Common vulnerabilities associated with insecure SCADA code design and implementation.

| Weakness Classification | Common Vulnerability |
|---|---|
| CWE-19: Data Handling | CWE-228: Improper Handling of Syntactically Invalid Structure |
| | CWE-229: Improper Handling of Values |
| | CWE-230: Improper Handling of Missing Values |
| | CWE-20: Improper Input Validation |
| | CWE-116: Improper Encoding or Escaping of Output |
| | CWE-195: Signed to Unsigned Conversion Error |
| | CWE-198: Use of Incorrect Byte Ordering |
| CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer | CWE-120: Buffer Copy without Checking Size of Input ("Classic Buffer Overflow") |
| | CWE-121: Stack-based Buffer Overflow |
| | CWE-122: Heap-based Buffer Overflow |
| | CWE-125: Out-of-bounds Read |
| | CWE-129: Improper Validation of Array Index |
| | CWE-131: Incorrect Calculation of Buffer Size |
| | CWE-170: Improper Null Termination |
| | CWE-190: Integer Overflow or Wraparound |
| | CWE-680: Integer Overflow to Buffer Overflow |
| CWE-398: Indicator of Poor Code Quality | CWE-454: External Initialization of Trusted Variables or Data Stores |
| | CWE-456: Missing Initialization |
| | CWE-457: Use of Uninitialized Variable |
| | CWE-476: NULL Pointer Dereference |
| | CWE-400: Uncontrolled Resource Consumption ("Resource Exhaustion") |
| | CWE-252: Unchecked Return Value |
| | CWE-690: Unchecked Return Value to NULL Pointer Dereference |
| | CWE-772: Missing Release of Resource after Effective Lifetime |
| CWE-442: Web Problems | CWE-22: Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal") |
| | CWE-79: Failure to Preserve Web Page Structure ("Cross-site Scripting") |
| | CWE-89: Failure to Preserve SQL Query Structure ("SQL Injection") |
| CWE-703: Failure to Handle Exceptional Conditions | CWE-431: Missing Handler |
| | CWE-248: Uncaught Exception |
| | CWE-755: Improper Handling of Exceptional Conditions |
| | CWE-390: Detection of Error Condition Without Action |

19




# DHS ICS Common Vulnerabilities…

assessments. Users may apply the tool to site-specific configurations, based on user created diagrams and selection of specific standards for each assessment.

CSET is a desktop software tool that guides users through a step-by-step question and answer process to collect facility-specific control and enterprise network information. The questions address topics such as hardware, software, administrative policies, and user obligations. After the user responds to the questions, the tool compares the information provided to relevant security standards and regulations, assesses overall compliance, and provides appropriate recommendations for improving the system's cybersecurity posture. The tool pulls its recommendations from a database of the best available cybersecurity practices, which have been adapted specifically for application to control system and enterprise networks and components. Where appropriate, recommendations are linked to a set of prioritized actions that can be applied to remediate specific security vulnerabilities.

CSET requirements were derived from widely accepted standards such as:

- DHS Catalog of Control Systems Security: Recommendations for Standards Development Revisions 4 and 6
- NIST SP 800-53: National Institute of Standards and Technology (NIST), Special Publication (SP) 800-53, Recommended Security Controls for Federal Information Systems, Revisions 0, 1, 2, and 3 Final Public Draft, June 2009
- NIST SP 800-82: National Institute of Standards and Technology, SP 800-82, Guide to Industrial Control Systems (ICS) Security, Final Public Draft, September 2008
- ISO/IEC 15408 (The Common Criteria): International Organization of Standards/ International Electrotechnical Commission, Version 3.1, September 2007
- DoDI 8500.2: US Department of Defense (DoD) Instruction Number 8500.2, "Information Assurance (IA) Implementation," February 6, 2003
- NERC CIP-002 through CIP-009: North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) (http://www.nerc.com/), Effective June 1, 2006.

#### 2.3.1.1 *Common CSET Findings*

The CSSP assisted in 50 CSET self-assessments in 2010 at owners and operations facilities within the 18 critical sectors, and in multiple CS2SAT self-assessments between 2006 and 2009. The CSSP provides the following benefits during the CSET evaluations:

- Cyber Security Awareness Briefing
- CSET training and demonstration
- "Over-the Shoulder" guidance to asset owners in using CSET
- Collective knowledge of common issues and good practices to identify vulnerabilities and mitigate risk
- Review assessment findings and provide mitigation techniques.

Table 4 summarizes the issues commonly identified as cybersecurity gap by ICS asset owners during onsite CSET assessments.

### 2.4 Compilation of ICS Vulnerability Information

DHS ICS risk reduction activities have gathered vulnerability information from many different types of ICS components, used by the multiple types of ICS. Information from different assessment approaches and ICS types provides a more complete picture of the security risks to ICS. Common types of vulnerabilities identified through CSSP assessments, ICS-CERT activities, and CSET self-assessments have been named and classified using consistent criteria, such as the Common Weakness Enumeration (CWE)[d] where possible, to enable correlation of vulnerability data. However, one should be careful about drawing conclusions from the data presented in this report.

d. http://cwe.mitre.org/

7

# Described using CWE…

Smart Grid Interchange Requirements Analysis and Standards Conformance Project

White Paper

# Automating Smart Grid Security

Applications of the Security Content Automation Protocol to the Smart Grid for Risk Management Activities

Date: December 7, 2011

Version: 1.4

Prepared for

National Institute of Standards and Technology

Under Contract Number: SB13110CN0101

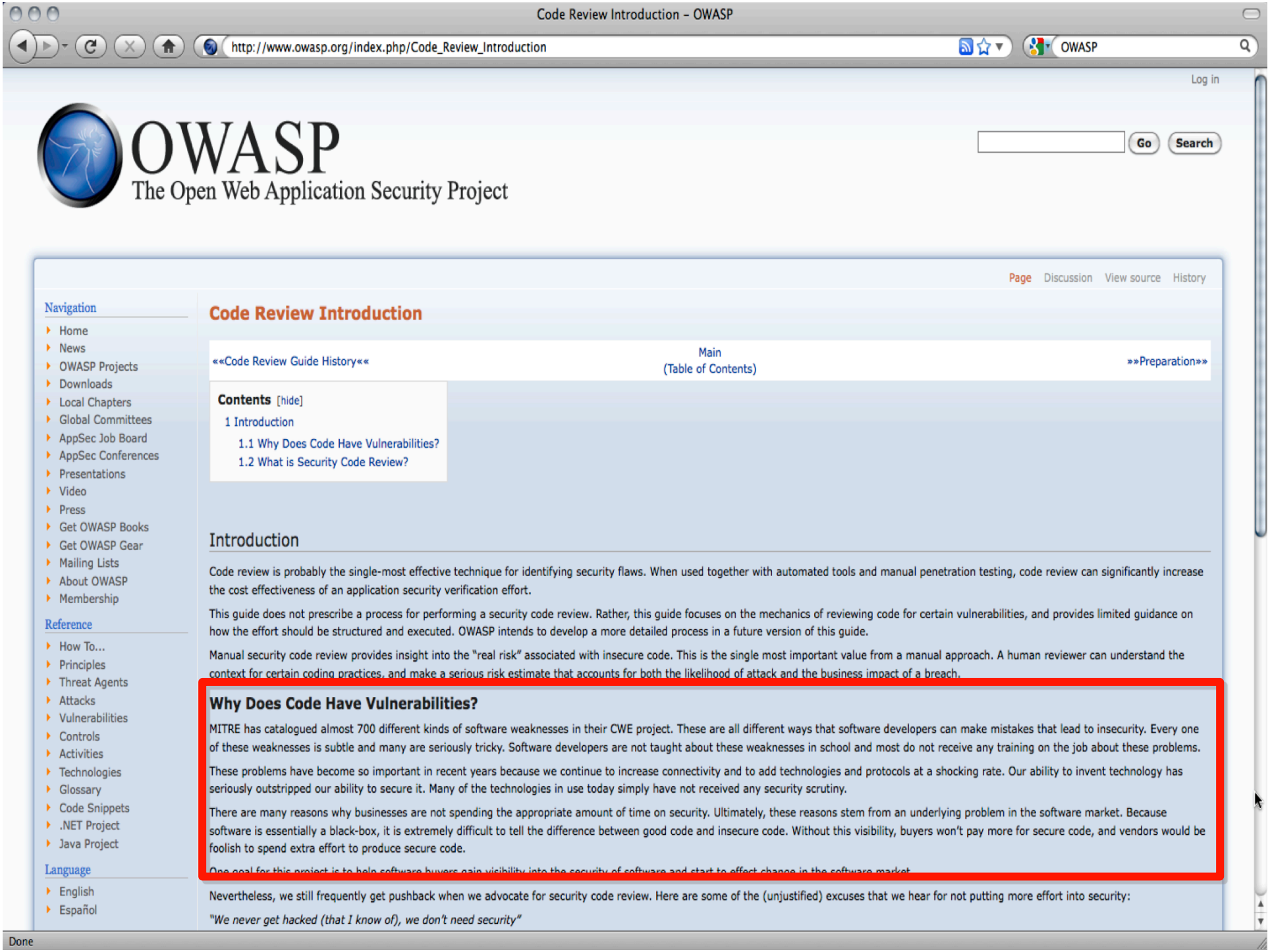
Then we provide a number of specific technical recommendations that could be undertaken to promote the use of SCAP to automate security processes in the Smart Grid and Industrial Control Systems more broadly. These recommendations include:

- Adopt the Asset Identification Format for Smart Grid Component Inventories
- Enhance ICS-CERT Security Advisories with Vulnerability Scoring
- Use of Asset Reporting Format for Interoperable Compliance Reporting
- Utilize Common Platform Enumeration
- Utilize Common Vulnerability Enumeration
- Extend OVAL Support to Smart Grid Systems
- Automate Smart Grid Continuous Monitoring
- Develop Security Checklists for Smart Grid Systems

## 5 Conclusions

The SCAP protocol was developed with the purpose of helping organizations maintain secure configurations, managing the technical aspects of assessing system compliance with security requirements, measuring security, automating security operations and communicating about vulnerabilities. These issues were identified in the IT security domain, but there is a considerable need to address these challenges in the Smart Grid and ICS domains as well. Security automation can help asset owners obtain a clear understanding of the security status of their systems, and SCAP provides a technical framework for doing this.

Including ICS related vulnerabilities in the CVE and ICS related platforms in the CPE is a simple first step towards utilization of SCAP in the Smart Grid and ICS domains. The use of CVSS for vulnerability scoring appears to be catching on. Additional applications such as development of XCCDF-expressed checklists and automated OVAL-based check systems are more challenging, but as the state of security awareness in the industry grows, it is hoped that the field of Smart Grid and ICS security can follow recent developments in the IT security domain to realize the benefits of security automation.


Code Review Introduction – OWASP
http://www.owasp.org/index.php/Code_Review_Introduction
OWASP
Log in
OWASP
The Open Web Application Security Project
Go
Search
Page
Discussion
View source
History
Navigation
Home
News
OWASP Projects
Downloads
Local Chapters
Global Committees
AppSec Job Board
AppSec Conferences
Presentations
Video
Press
Get OWASP Books
Get OWASP Gear
Mailing Lists
About OWASP
Membership
Reference
How To...
Principles
Threat Agents
Attacks
Vulnerabilities
Controls
Activities
Technologies
Glossary
Code Snippets
.NET Project
Java Project
Language
English
Español
Code Review Introduction
««Code Review Guide History««
Main
(Table of Contents)
»»Preparation»»
Contents [hide]
1 Introduction
1.1 Why Does Code Have Vulnerabilities?
1.2 What is Security Code Review?
Introduction
Code review is probably the single-most effective technique for identifying security flaws. When used together with automated tools and manual penetration testing, code review can significantly increase the cost effectiveness of an application security verification effort.
This guide does not prescribe a process for performing a security code review. Rather, this guide focuses on the mechanics of reviewing code for certain vulnerabilities, and provides limited guidance on how the effort should be structured and executed. OWASP intends to develop a more detailed process in a future version of this guide.
Manual security code review provides insight into the "real risk" associated with insecure code. This is the single most important value from a manual approach. A human reviewer can understand the context for certain coding practices, and make a serious risk estimate that accounts for both the likelihood of attack and the business impact of a breach.
Why Does Code Have Vulnerabilities?
MITRE has catalogued almost 700 different kinds of software weaknesses in their CWE project. These are all different ways that software developers can make mistakes that lead to insecurity. Every one of these weaknesses is subtle and many are seriously tricky. Software developers are not taught about these weaknesses in school and most do not receive any training on the job about these problems.
These problems have become so important in recent years because we continue to increase connectivity and to add technologies and protocols at a shocking rate. Our ability to invent technology has seriously outstripped our ability to secure it. Many of the technologies in use today simply have not received any security scrutiny.
There are many reasons why businesses are not spending the appropriate amount of time on security. Ultimately, these reasons stem from an underlying problem in the software market. Because software is essentially a black-box, it is extremely difficult to tell the difference between good code and insecure code. Without this visibility, buyers won't pay more for secure code, and vendors would be foolish to spend extra effort to produce secure code.
Nevertheless, we still frequently get pushback when we advocate for security code review. Here are some of the (unjustified) excuses that we hear for not putting more effort into security:
"We never get hacked (that I know of), we don't need security"
Done

# Threat Classification Taxonomy Cross Reference View

last edited by Robert Auger 10 months, 3 weeks ago

Page history

Tags: Threat Classification

Check for plagiarism

## Threat Classification 'Taxonomy Cross Reference View'

This view contains a mapping of the WASC Threat Classification's Attacks and Weaknesses with MITRE's Common Weakness Enumeration, MITRE's Common Attack Pattern Enumeration and Classification, OWASP Top Ten 2010 RC1 (original mapping with OWASP Top Ten from Jeremiah Grossman & Bill Corry) and SANS/CWE and OWASP Top Ten 2007 and 2004 (original mapping from Dan Cornell, Denim Group)

| WASC ID | Name | CWE ID | CAPEC ID | SANS/CWE Top 25 2009 | OWASP Top Ten 2010 | OWASP Top Ten 2007 | OWASP Top Ten 2004 |
|---|---|---|---|---|---|---|---|
| WASC-01 | Insufficient Authentication | 287 | | 642 | A3 – Broken Authentication and Session Management, A4 – Insecure Direct Object References | A7 – Broken Authentication and Session Management, A4 – Insecure Direct Object Reference | A3 – Broken Authentication and Session management, A2 – Broken Access Control |
| WASC-02 | Insufficient Authorization | 284 | | 285 | A4 – Insecure Direct Object References, A7 – Failure to Restrict URL Access | A10 – Failure to Restrict URL Access, A4 – Insecure Direct Object Reference | A2 – Broken Access Control |
| WASC-03 | Integer Overflows | 190 | 128 | 682 | | | |
| WASC-04 | Insufficient Transport Layer Protection | 311 523 | | 319 | A10 – Insufficient Transport Layer Protection | A9 – Insecure Communications | |
| WASC-05 | Remote File Inclusion | 98 | 193 253 | 426 | | A3 – Malicious File Execution | |
| WASC-06 | Format String | 134 | 67 | | | | |
| WASC-07 | Buffer Overflow | 119 120 | 10 100 | 119 | | | A5 – Buffer Overflows |
| WASC-08 | Cross-site Scripting | 79 | 18 19 63 | 79 | A2 – Cross-Site Scripting | A1 – Cross Site Scripting (XSS) | A4 – Cross Site Scripting (XSS) |
| WASC-09 | Cross-site Request Forgery | 352 | 62 | 352 | A5 – Cross-Site Request Forgery | A5 – Cross Site Request Forgery (CSRF) | |
| WASC-10 | Denial of Service | 400 | 119 | 404 | A7 – Failure to Restrict | A10 – Failure to | A9 – Denial of |

SideBar

WASC Projects
- Distributed Open Proxy Honeypots
- Script Mapping
- The Web Security Glossary
- Web Application Firewall Evaluation Criteria
- Web Application Security Scanner Evaluation Criteria
- Web Application Security Statistics
- Web Hacking Incidents Database
- WASC Threat Classification

WASC Project Leaders
- Robert Auger
- Ryan Barnett
- Romain Gaucher
- Sergey Gordeychik
- Ofer Shezaf
- Brian Shura

WASC Main Website
- http://www.webappsec.org/

WASC Mailing Lists
- http://lists.webappsec.org/

WASC on Twitter
- http://twitter.com/wascupdates

Join us on Linkedin!
- http://www.linkedin.com/groups?qid=83336

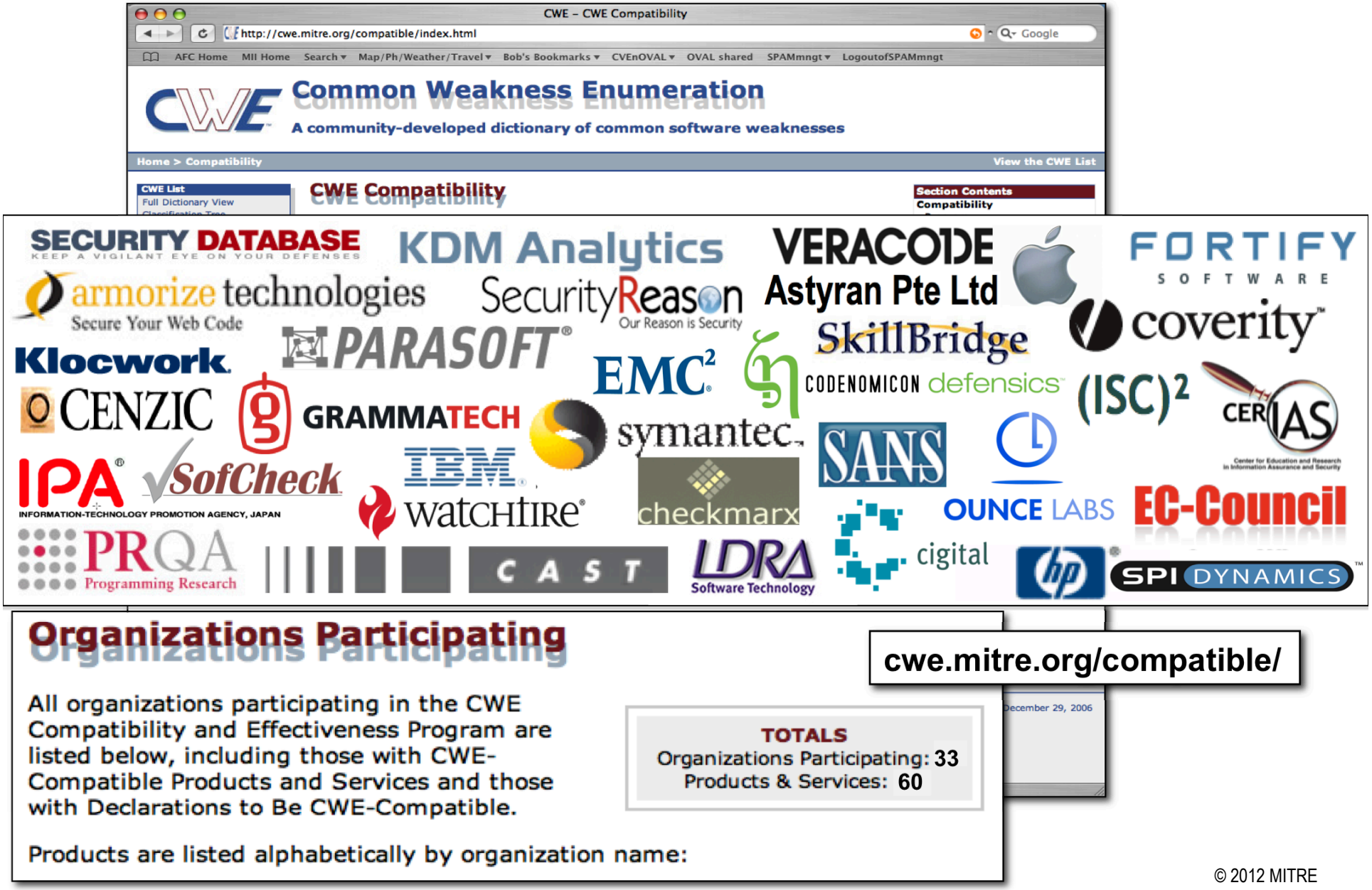
Recent Activity

Insufficient Data Protection Working
edited by Robert Auger

# CWE web site visitors by City

# CWE Compatibility & Effectiveness Program

**( launched Feb 2007)**

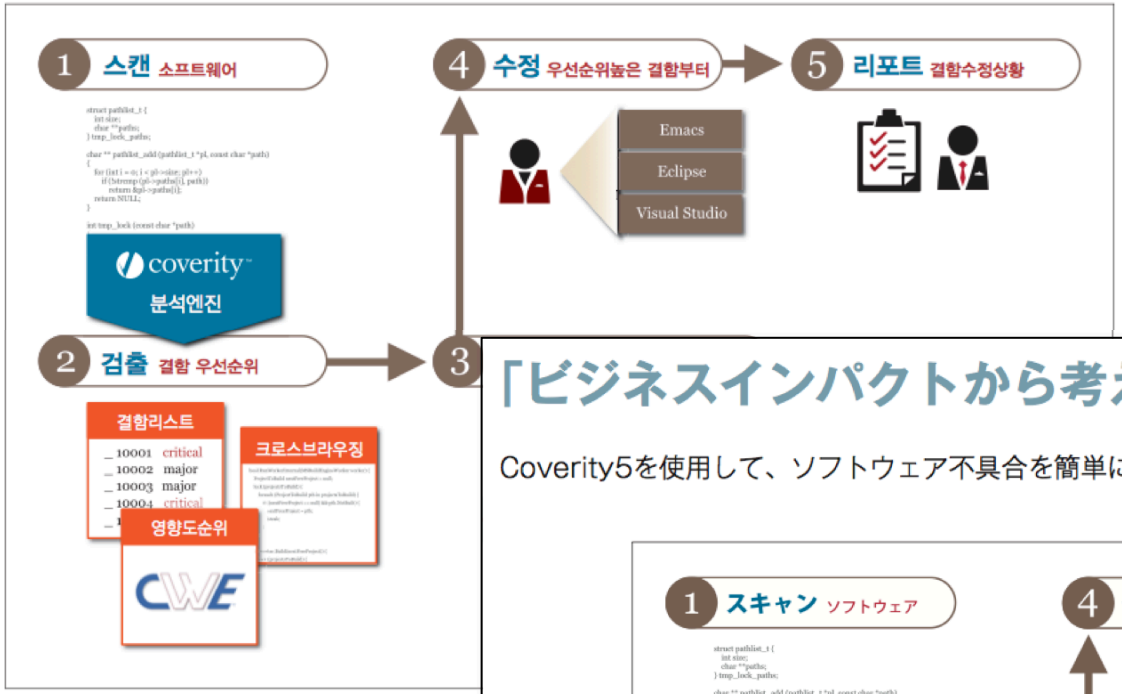



**cwe.mitre.org/compatible/**


armorize
The Web Malware Experts
We speak
CWE
cwe.mitre.org
We speak
CVE
cve.mitre.org
GLOBAL INFORMATION ASSURANCE CERTIFICATION
GIAC
SANS
coverity
Coverity + Armorize
int val; };
my_struct *ptr, int x, int y)
) && (x < y) || (ptr->val > 0))
tr->val;

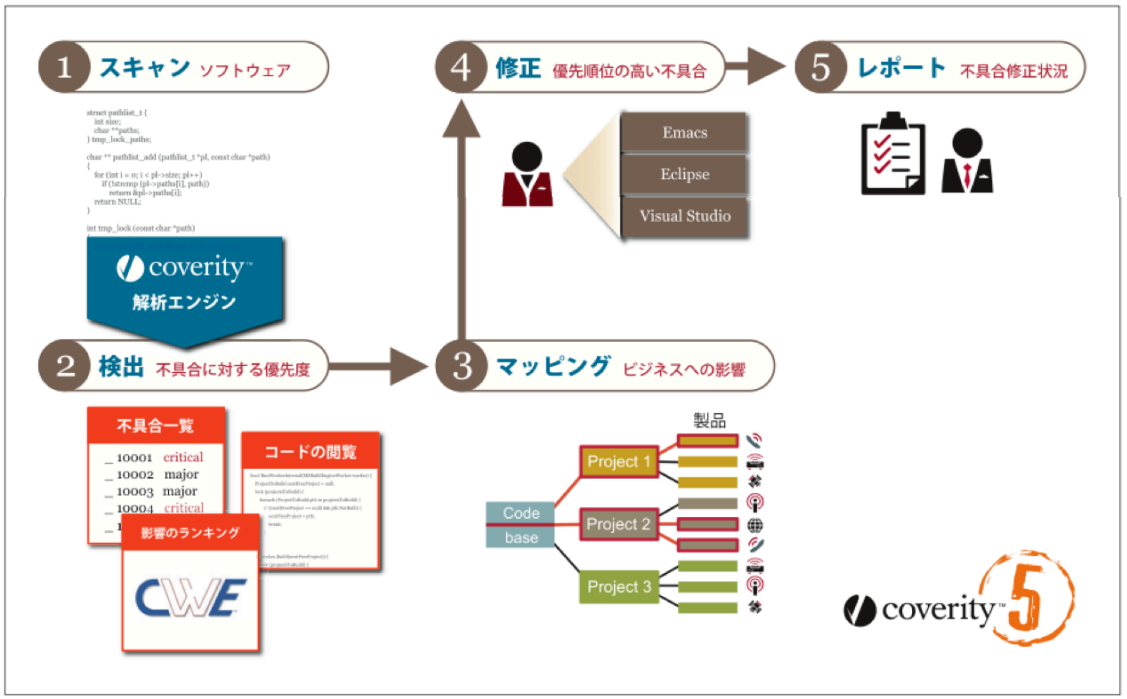
## [비즈니스 임팩트를 줄여주는 새로운 품질 관리 방법론]

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




## 「ビジネスインパクトから考える新しい品質管理」

Coverity5を使用して、ソフトウェア不具合を簡単に除去する 5ステップは以下の通りです。




**Korean**

**Japanese**

# CWE Coverage – Implemented…

Coverity® Data Sheet

coverity

## Coverity Coverage for Common Weakness Enumeration (CWE): Java

| CWE ID | Coverity Static Analysis Checker |
|---|---|
| 171 | BAD_EQ |
| 252 | CHECKED_RETURN |
| 366 | GUARDED_BY_VIOLATION |
| | INDIRECT_GUARDED_BY_VIOLATION |
| | NON_STATIC_GUARDING_STATIC |
| | VOLATILE_ATOMICITY |
| 382 | DC.CODING_STYLE |
| | BAD_OVERRIDE |
| | DC.EXPLICIT_DEPRECATION |
| | DC.GC |
| | MUTABLE_COMPARISON |
| 390 | MUTABLE_HASHCODE |

Coverity® Data Sheet

coverity

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

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

CENZIC

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

### Cenzic Product Suite is CWE Compatible

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

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

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

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







## CWE IDs mapped to Klocwork Java issue types

**From current**

CWE IDs mapped to Klocwork Java issue types

See also Detected Java Issues.

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





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

**From current**

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

その他の情報 Detected C and C++ Issues.

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

Stockhouse

Taking it t

U.S. Rolls Out Pla

US rolls out plan to protect busi

COMPUTERW

The voice of the ICT community

HOME NEWS TECHNOLOG

TUESDAY, 28 JUNE 2011

LATEST NEWS

ASB Bank, Potentia and Hairy Lemon pick up 2011 CIO awards

IT and marketing are a killer

Forbes.com

U.S. EUROPE ASIA

Home Business Investin

Video ForbesWoman CEO Network

Associated Press

US rolls out plan to pr

LOLITA C. BALDOR , 06.27.11

AMERICAN PUBLIC MEDIA

Marketplace®

Contact | About | Local Air Times | Newsletters | Su

Shows Sections Topics Podcasts

Department of Homeland Security wants to help you pro

By John Moe

Marketplace Tech

If you have a

about security

Home : Sci-Tech : U.S. launches plan to protect business websites

U.S. launches plan to protect business websites

View Larger Image

A A | Email | Print

Recommend Sign Up to see what your friends recommend.

The Associated Press

Date: Monday Jun. 27, 2011 9:10 PM ET

WASHINGTON — American businesses facing a growing threat of cyberattacks against their websites now will have more tools to protect themselves and harden their Internet sites against hackers.

The Washington Times

U.S. rolls

0 Comments

HOME

NECN.com

73°

CANADIAN BUSINESS

Home News & Markets Blogs & Analysis

HOT TOPICS: Leadership Q&A Retirement Business Briefings

Topics News & Markets

DHS rolls out plan to help protect small from hackers

By Lolita C. Baldor, The Associated Press | June 27, 2011

Like 2 Share

WASHINGTON - Businesses facing a growing threat of cyberattacks more tools to protect themselves and harden their Internet sites against hackers.

Most emaile

- Some Fl urgent-c will be re post pric common
- Most exp home in Hillsboro estate m got a litt
- Convicte driver wi license a Tampa

Print

Email

Save

used in an SQ

Hackers used security compa

MITRE

SANS

| Rank | Score | ID | Name | | |
|---|---|---|---|---|---|
| [1] | 93.8 | CWE-89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') | up from 2 | +1 |
| [2] | 83.3 | CWE-78 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') | up from 9 | +7 |
| [3] | 79.0 | CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') | same | 0 |
| [4] | 77.7 | CWE-79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') | down from 1 | -3 |
| [5] | 76.9 | CWE-306 | Missing Authentication for Critical Function | up from 19 | +14 |
| [6] | 76.8 | CWE-862 | Missing Authorization | split of prior #5 | -1 |
| [7] | 75.0 | CWE-798 | Use of Hard-coded Credentials | up from 11 | +4 |
| [8] | 75.0 | CWE-311 | Missing Encryption of Sensitive Data | up from 10 | +2 |
| [9] | 74.0 | CWE-434 | Unrestricted Upload of File with Dangerous Type | down from 8 | -1 |
| [10] | 73.8 | CWE-807 | Reliance on Untrusted Inputs in a Security Decision | down from 6 | -4 |
| [11] | 73.1 | CWE-250 | Execution with Unnecessary Privileges | new entry | n/a |
| [12] | 70.1 | CWE-352 | Cross-Site Request Forgery (CSRF) | down from 4 | -8 |
| [13] | 69.3 | CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') | down from 7 | -6 |
| [14] | 68.5 | CWE-494 | Download of Code Without Integrity Check | up from 20 | +6 |
| [15] | 67.8 | CWE-863 | Incorrect Authorization | split of prior #5 | -10 |


Homeland Security

TOP 25

MITRE

SANS

| | | | | | |
|---|---|---|---|---|---|
| [16] | 66.0 | CWE-829 | Inclusion of Functionality from Untrusted Control Sphere | new entry | n/a |
| [17] | 65.5 | CWE-732 | Incorrect Permission Assignment for Critical Resource | up from 21 | +4 |
| [18] | 64.6 | CWE-676 | Use of Potentially Dangerous Function | new entry | n/a |
| [19] | 64.1 | CWE-327 | Use of a Broken or Risky Cryptographic Algorithm | up from 24 | +5 |
| [20] | 62.4 | CWE-131 | Incorrect Calculation of Buffer Size | down from 18 | -2 |
| [21] | 61.5 | CWE-307 | Improper Restriction of Excessive Authentication Attempts | new entry | n/a |
| [22] | 61.1 | CWE-601 | URL Redirection to Untrusted Site ('Open Redirect') | up from 23 | +1 |
| [23] | 61.0 | CWE-134 | Uncontrolled Format String | new entry | n/a |
| [24] | 60.3 | CWE-190 | Integer Overflow or Wraparound | down from 17 | -7 |
| [25] | 59.9 | CWE-759 | Use of a One-Way Hash without a Salt | new entry | n/a |


Homeland Security

TOP 25

MITRE

SANS

## Insecure Interaction Between Components

| Rank | CWE ID | Name |
|---|---|---|
| [1] | CWE-89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') |
| [2] | CWE-78 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') |
| [4] | CWE-79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') |
| [9] | CWE-434 | Unrestricted Upload of File with Dangerous Type |
| [12] | CWE-352 | Cross-Site Request Forgery (CSRF) |
| [22] | CWE-601 | URL Redirection to Untrusted Site ('Open Redirect') |

## Risky Resource Management

| Rank | CWE ID | Name |
|---|---|---|
| [3] | CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') |
| [13] | CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') |
| [14] | CWE-494 | Download of Code Without Integrity Check |
| [16] | CWE-829 | Inclusion of Functionality from Untrusted Control Sphere |
| [18] | CWE-676 | Use of Potentially Dangerous Function |
| [20] | CWE-131 | Incorrect Calculation of Buffer Size |
| [23] | CWE-134 | Uncontrolled Format String |
| [24] | CWE-190 | Integer Overflow or Wraparound |

## Porous Defenses

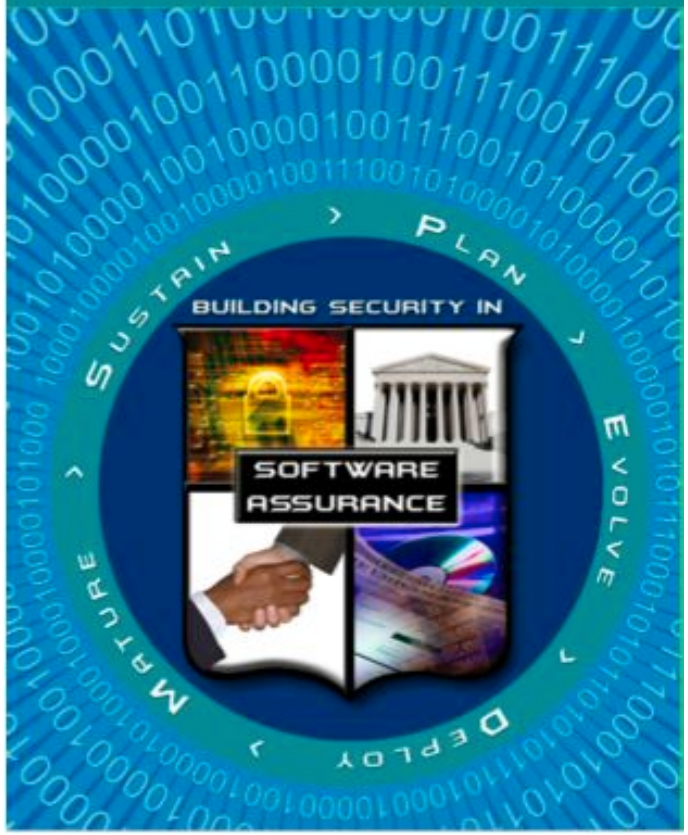
| Rank | CWE ID | Name |
|---|---|---|
| [5] | CWE-306 | Missing Authentication for Critical Function |
| [6] | CWE-862 | Missing Authorization |
| [7] | CWE-798 | Use of Hard-coded Credentials |
| [8] | CWE-311 | Missing Encryption of Sensitive Data |
| [10] | CWE-807 | Reliance on Untrusted Inputs in a Security Decision |
| [11] | CWE-250 | Execution with Unnecessary Privileges |
| [15] | CWE-863 | Incorrect Authorization |
| [17] | CWE-732 | Incorrect Permission Assignment for Critical Resource |
| [19] | CWE-327 | Use of a Broken or Risky Cryptographic Algorithm |
| [21] | CWE-307 | Improper Restriction of Excessive Authentication Attempts |
| [25] | CWE-759 | Use of a One-Way Hash without a Salt |



TOP 25

# Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses

Software Assurance Pocket Guide Series:
Development, Volume II
Version 2.2, June 26, 2012 (Draft)




## Software Assurance (SwA) Pocket Guide Resources

This is a resource for 'getting started' in selecting and adopting relevant practices for engineering, developing, and delivering secure software. As part of the Software Assurance (SwA) Pocket Guide series, this resource is offered for informative use only; it is not intended as directive or presented as being comprehensive since it references and summarizes material in the source documents and on-line resources that provide detailed information. When referencing any part of this document, please provide proper attribution and reference the source documents, when applicable.

*This volume of the SwA Pocket Guide series focuses on key practices for mitigating the most egregious exploitable software weaknesses. It identifies mission/business risks attributable to the respective weaknesses, it identifies common attacks that exploit those weaknesses, and provides recommended practices for preventing the weaknesses. It provides insight for how software weaknesses are prioritized to guide training, development and procurement efforts.*

At the back of this pocket guide are references, limitation statements, and a listing of topics addressed in the SwA Pocket Guide series. All SwA Pocket Guides and SwA-related documents are freely available for download via the SwA Community Resources and Information Clearinghouse at http://buildsecurityin.us-cert.gov/swa .


## Acknowledgements

The SwA Forum and Working Groups function as a stakeholder mega-community that welcomes additional participation in advancing software security and refining SwA-related information resources that are offered free for public use. Input to all SwA resources is encouraged. Please contact Software.Assurance@dhs.gov for comments and inquiries.

The SwA Forum is composed of government, industry, and academic members. The SwA Forum focuses on incorporating SwA considerations in educatoin, acquisition, and development processes relative to potential risk exposures that could be introduced by software and the software supply chain.

Participants in the SwA Forum's Processes & Practices Working Group collaborated with the Technology, Tools and Product Evaluation Working Group in developing the material used in this pocket guide as a step in raising awareness on how to incorporate SwA throughout the Software Development Life Cycle (SDLC).

Lacking common characterization of exploitable software constructs and how they could be attacked with associated mitigation practices previously presented one of the major challenges to realizing software assurance objectives. As part

- *"Fundamental Practices for Secure Software Development, 2ND EDITION, A Guide to the Most Effective Secure Development Practices in Use Today", SAFECode, February 8, 2011 at http://www.safecode.org/publications/SAFECode_Dev_Practices0211.pdf*

## Background

The 2011 CWE/SANS Top 25 Most Dangerous Programming Errors is a consensus list of the most significant programming errors that can lead to serious software vulnerabilities. They occur frequently, are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.

The list is the result of collaboration between the MITRE CWE team, many top software security experts in the US and Europe, and the SANS Institute. It leverages experiences in the development of the SANS Top 20 attack vectors ( http://www.sans.org/top20/ ), MITRE's Common Weakness Enumeration (CWE) ( http://cwe.mitre.org/ ), and MITRE's Common Attack Pattern Enumeration and Classification (CAPEC) ( https://capec.mitre.org/ ). With the sponsorship and support of the US Department of Homeland Security's National Cyber Security Division Software Assurance Program, MITRE maintains the CWE and CAPEC websites, presenting detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigating and avoiding them. The CWE site also contains data on more than 800 additional programming errors, design errors, and architecture errors that can lead to exploitable vulnerabilities. See CWE Frequently Asked Questions at http://cwe.mitre.org/about/faq.html .

A goal for the CWE Top 25 list is to stop vulnerabilities at the source by educating programmers on how to eliminate all-too-common mistakes before software is even shipped. The list serves as a tool for education and awareness to help programmers prevent the kinds of vulnerabilities that plague the software industry. Software consumers can use the same list to help them to ask for more secure software. Finally, software managers, testers, and CIOs can use the CWE Top 25 list as a means for selecting the best tools and services for their needs and as a measuring stick of progress in their efforts to secure their software.

## Top 25 Common Weaknesses

Table 1 provides the Top 25 CWEs organized into three high-level categories that contain multiple CWE entries:

1. Insecure Interaction Between Components
2. Risky Resource Management
3. Porous Defenses

*Table 1 - Top 25 Common Weakness Enumeration (CWE)*

| 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** | **Description** |
| CWE-78 | Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'). |
| CWE-79 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'). |
| CWE-89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). |
| CWE-352 | Cross-Site Request Forgery (CSRF). |
| CWE-434 | Unrestricted Upload of File with Dangerous Type. |
| CWE-601 | URL Redirection to Untrusted Site ('Open Redirect'). |



| Risky Resource Management | |
|---|---|
| These weaknesses are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources. | |
| **CWE** | **Description** |
| CWE-22 | Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'). |
| CWE-120 | Buffer Copy without Checking Size of Input ('Classic Buffer Overflow'). |
| CWE-131 | Incorrect Calculation of Buffer Size. |
| CWE-134 | Uncontrolled Format String. |
| CWE-190 | Integer Overflow or Wraparound. |
| CWE-494 | Download of Code Without Integrity Check. |
| CWE-676 | Use of Potentially Dangerous Function. |
| CWE-829 | Inclusion of Functionality from Untrusted Control Sphere. |
| **Porous Defenses** | |
| These weaknesses are related to defensive techniques that are often misused, abused, or just plain ignored. | |
| **CWE** | **Description** |
| CWE-250 | Execution with Unnecessary Privileges. |
| CWE-306 | Missing Authentication for Critical Function. |
| CWE-307 | Improper Restriction of Excessive Authentication Attempts. |
| CWE-311 | Missing Encryption of Sensitive Data. |
| CWE-327 | Use of a Broken or Risky Cryptographic Algorithm. |
| CWE-732 | Incorrect Permission Assignment for Critical Resource. |
| CWE-759 | Use of a One-Way Hash without a Salt. |
| CWE-798 | Use of Hard-coded Credentials. |
| CWE-807 | Reliance on Untrusted Inputs in a Security Decision. |
| CWE-862 | Missing Authorization. |
| CWE-863 | Incorrect Authorization. |

## Selection of the Top 25 CWEs

The Top 25 CWE list was first developed at the end of 2008 and is updated on a yearly basis. Approximately 40 software security experts provided feedback, including software developers, scanning tool vendors, security consultants, government representatives, and university professors. Representation was international. Intermediate versions were created and resubmitted to the reviewers before the list was finalized. More details are provided in the Top 25 Process page at http://cwe.mitre.org/top25/process.html .

To help characterize and prioritize entries in the Top 25 CWE list, a threat model was developed that identified an attacker with solid technical skills and determined enough to invest some time into attacking an organization. Weaknesses in the Top 25 were selected using two primary criteria:

- **Weakness Prevalence:** how often the weakness appears in software that was not developed with security integrated into the software development life cycle (SDLC).
- **Consequences:** the typical consequences of exploiting a weakness if it is present, such as unexpected code execution, data loss, or denial of service.

Prevalence was determined based on estimates from multiple contributors to the Top 25 list, since appropriate statistics were not readily available.

With these criteria, future versions of the Top 25 CWEs will evolve to cover different weaknesses. Other CWEs that represent significant risks were listed as being on the cusp, and they can be viewed at http://cwe.mitre.org/ .

## Information about the Weaknesses

The primary audience for CWE information is intended to be software programmers and designers. For each individual CWE entry, additional information is provided.

- CWE ID and name.
- Supporting data fields: supplementary information about the weakness that may be useful for decision-makers to further prioritize the entries.
- Discussion: Short, informal discussion of the nature of the weakness and its consequences.
- Prevention and Mitigations: steps that developers can take to mitigate or eliminate the weakness. Developers may choose one or more of these mitigations to fit their own needs. Note that the effectiveness of these techniques vary, and multiple techniques may be combined for greater defense-in-depth.
- Related CWEs: other CWE entries that are related to the Top 25 weakness. Note: This list is illustrative, not comprehensive.
- Related Attack Patterns: CAPEC entries for attacks that may be successfully conducted against the weakness. Note: the list is not necessarily complete.

See http://cwe.mitre.org for the additional supporting information on each CWE.

## Other Supporting Data Fields in CWEs

Each Top 25 entry includes supporting data fields for weakness prevalence and consequences. Each entry also includes the following data fields.

- **Attack Frequency:** how often the weakness occurs in vulnerabilities that are exploited by an attacker.
- **Ease of Detection:** how easy it is for an attacker to find this weakness.
- **Remediation Cost:** the amount of effort required to fix the weakness.
- **Attacker Awareness:** the likelihood that an attacker is going to be aware of this particular weakness, methods for detection, and methods for exploitation.

## Associated Mission/Business Risks and Related Attack Patterns

For each common weakness in software, there are associated risks to the mission or business enabled by the software. Moreover, there are common attack patterns that exploit those weaknesses.

Attack patterns are powerful mechanisms that capture and communicate the attacker's perspective. They are descriptions of common methods for exploiting software. They derive from the concept of design patterns applied in a destructive rather than constructive context and are generated from in-depth analysis of specific real-world exploit examples. To assist in enhancing security throughout the software development lifecycle, and to support the needs of developers, testers and educators, the **CWE and Common Attack Pattern Enumeration and Classification (CAPEC)** are co-sponsored by DHS National Cyber Security Division as part of the Software Assurance strategic initiative, and the efforts are managed by MITRE. The CAPEC website provides a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy. CAPEC will continue to evolve with public participation and contributions to form a standard mechanism for identifying, collecting, refining, and sharing attack patterns among the software community.



Development teams should use attack patterns to understand the resilience of their software relative to common attacks and misuse. Table 2 lists the Mission/Business risks associated with each CWE, and it lists some of the possible attacks and misuses associated with the relevant CWEs which enable exploitation of the software.

For a full listing and description of all the attacks related to a particular CWE visit the websites for CWE and CAPEC at http://cwe.mitre.org and http://capec.mitre.org .

*Table 2 - CWEs and Their Related Attack Patterns and Mission/Business Risks*

| CWE | Related Attack Pattern | Mission/Business Risks |
|---|---|---|
| CWE-22 : Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') | » CAPEC-23: File System Function Injection, Content Based<br>» CAPEC-64: Using Slashes and URL Encoding Combined to Bypass Validation Logic<br>» CAPEC-76: Manipulating Input to File System Calls<br>» CAPEC-78: Using Escaped Slashes in Alternate Encoding<br>» CAPEC-79: Using Slashes in Alternate Encoding<br>» CAPEC-139: Relative Path Traversal | » DoS: crash / exit / restart<br>» Execute unauthorized code or commands<br>» Modify files or directories<br>» Read files or directories |
| CWE-78 : Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') | » CAPEC-6: TCP Header<br>» CAPEC-15: Command Delimiters<br>» CAPEC-43: Exploiting Multiple Input Interpretation Layers<br>» CAPEC-88: OS Command Injection<br>» CAPEC-108: Command Line Execution through SQL Injection | » DoS: crash / exit / restart<br>» Execute unauthorized code or commands<br>» Hide activities<br>» Modify application data<br>» Modify files or directories<br>» Read application data<br>» Read files or directories |
| CWE-79 : Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') | » CAPEC-18: Embedding Scripts in Nonscript Elements<br>» CAPEC-19: Embedding Scripts within Scripts<br>» CAPEC-32: Embedding Scripts in HTTP Query Strings<br>» CAPEC-63: Simple Script Injection<br>» CAPEC-85: Client Network Footprinting (using AJAX/XSS)<br>» CAPEC-86: Embedding Script (XSS ) in HTTP Headers<br>» CAPEC-91: XSS in IMG Tags<br>» CAPEC-106: Cross Site Scripting through Log Files<br>» CAPEC-198: Cross-Site Scripting in Error Pages<br>» CAPEC-199: Cross-Site Scripting Using Alternate Syntax<br>» CAPEC-209: Cross-Site Scripting Using MIME Type Mismatch<br>» CAPEC-232: Exploitation of Privilege/Trust<br>» CAPEC-243: Cross-Site Scripting in Attributes<br>» CAPEC-244: Cross-Site Scripting via Encoded URI Schemes | » Bypass protection mechanism<br>» Execute unauthorized code or commands<br>» Read application data |

| Table 2 - CWEs and Their Related Attack Patterns and Mission/Business Risks | | |
|---|---|---|
| **CWE** | **Related Attack Pattern** | **Mission/Business Risks** |
| | » CAPEC-184: Software Integrity Attacks<br>» CAPEC-185: Malicious Software Download<br>» CAPEC-193: PHP Remote File Inclusion<br>» CAPEC-222: iFrame Overlay<br>» CAPEC-251: Local Code Inclusion<br>» CAPEC-252: PHP Local File Inclusion<br>» CAPEC-253: Remote Code Inclusion | |
| CWE-862 : Missing Authorization | » CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs<br>» CAPEC-17: Accessing, Modifying or Executing Executable Files<br>» CAPEC-58: Restful Privilege Elevation<br>» CAPEC-122: Exploitation of Authorization<br>» CAPEC-180: Exploiting Incorrectly Configured Access Control Security Levels | » Bypass protection mechanism<br>» Gain privileges / assume identity<br>» Modify application data<br>» Modify files or directories<br>» Read application data<br>» Read files or directories |
| CWE-863 : Incorrect Authorization | » CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs<br>» CAPEC-17: Accessing, Modifying or Executing Executable Files<br>» CAPEC-58: Restful Privilege Elevation<br>» CAPEC-122: Exploitation of Authorization<br>» CAPEC-180: Exploiting Incorrectly Configured Access Control Security Levels | » Bypass protection mechanism<br>» Gain privileges / assume identity<br>» Modify application data<br>» Modify files or directories<br>» Read application data<br>» Read files or directories |

## Key Practices

The key practices documented in "2011 CWE/SANS Top 25 Most Dangerous Programming Errors" focus on preventing and mitigating dangerous programming errors. Some of the Key Practices specified in the pocket guide are derived from mitigation recommendations that were common across many of the CWEs in the CWE Top 25, and others came from approaches described on the CERT Secure Coding Wiki. Additional information on preventing the various weaknesses is available in the CERT Secure Coding Wiki at https://www.securecoding.cert.org/ and other websites listed under On-Line Resources of this SwA Pocket Guide. Development teams are also encouraged to use the CAPEC attack patterns to gain understanding of how their software can be attacked, as well as considering how they can engineer their software to better handle such attacks. They are also encouraged to use the CAPEC attack patterns to develop tests that can determine the resilience of their code relative to the common attacks used to exploit software weaknesses. In this SwA Pocket Guide the key practices are grouped in tables according to Software Development Life Cycle (SDLC) phases:

1. Requirements, Architecture, and Design (Table 3) ;
2. Build, Compilation, Implementation, Testing, and Documentation (Table 4) ;
3. Installation, Operation and System Configuration (Table 5) , and
4. Associated CERT Coding Rules (Table 6) .

| Table 3 – Requirements, Architecture, and Design | |
|---|---|
| **Prevention and Mitigation Practices** | **CWE** |
| For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. | CWE-22 : Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') |
| Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. | |
| When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.<br>For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability.[R.22.3] | |
| If at all possible, use library calls rather than external processes to recreate the desired functionality. | CWE-78 : Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') |
| For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the data locally in the session's state instead of sending it out to the client in a hidden form field. | |
| For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. | |
| Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.<br>For example, consider using the ESAPI Encoding control [R.78.8] or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error. | |
| 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.<br>Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments. | |

*Table 6 – Associated CERT Coding Rules*

| Prevention and Mitigation Practices | CWE |
|---|---|
| No associated CERT coding rules listed for this CWE entry. | CWE-434 : Unrestricted Upload of File with Dangerous Type |
| SEC06-J: Do not rely on the default automatic signature verification provided by | CWE-494 : Download of Code Without Integrity Check |
| No associated CERT coding rules listed for this CWE entry. | CWE-601 : URL Redirection to Untrusted Site ('Open Redirect') |
| ERR07-C: Prefer functions that support error checking over equivalent functions that | CWE-676 : Use of Potentially Dangerous Function |
| FIO01-C: Be careful using functions that use file names for identification | |
| INT06-C: Use strtol() or a related function to convert a string token to an integer | |
| INT06-CPP: Use strtol() or a related function to convert a string token to an integer | |
| FIO01-CPP: Be careful using functions that use file names for identification | |
| FIO03-J: Create files with appropriate access permission | CWE-732 : Incorrect Permission Assignment for Critical Resource |
| SEC01-J: Do not allow tainted variables in privileged blocks | |
| ENV03-J: Do not grant dangerous combinations of permissions | |
| FIO06-CPP: Create files with appropriate access permissions | |
| FIO06-C: Create files with appropriate access permissions | |
| No associated CERT coding rules listed for this CWE entry. | CWE-759 : Use of a One-Way Hash without a Salt |
| MSC03-J: Never hard code sensitive information | CWE-798 : Use of Hard-coded Credentials |
| ENV03-CPP: Sanitize the environment when invoking external programs | CWE-807 : Reliance on Untrusted Inputs in a Security Decision |
| SEC09-J: Do not base security checks on untrusted sources | |
| No associated CERT coding rules listed for this CWE entry. | CWE-829 : Inclusion of Functionality from Untrusted Control Sphere |
| No associated CERT coding rules listed for this CWE entry. | CWE-862 : Missing Authorization |
| No associated CERT coding rules listed for this CWE entry. | CWE-863 : Incorrect Authorization |

*Table 7 - Shared Mitigations*

| Mitigation | CWE Entries |
|---|---|
| MIT-10 | Run or compile your software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.<br>For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio /GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.<br>1. CWE-120 : Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')<br>2. CWE-131 : Incorrect Calculation of Buffer Size |
| MIT-11 | Use a feature like Address Space Layout Randomization (ASLR).[R.XX.A] [R.XX.B] |



*Table 7 - Shared Mitigations*

| Mitigation | CWE Entries |
|---|---|
| | 1. CWE-131 : Incorrect Calculation of Buffer Size<br>2. CWE-190 : Integer Overflow or Wraparound |
| MIT-9 | Consider adhering to the following rules when allocating and managing an application's memory:<br>1. CWE-120 : Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') |

## Creating Custom Top-N Lists using CWSS and CWRAF

The CWE/SANS Top 25 is a great starting point, but each organization has its own set of business priorities, threat environment, and risk tolerance and for those with the an understanding of those issues, a more refined and custom Top 25 for their business and what software is doing for their business is possible through the Common Weakness Scoring System (CWSS) (https://cwe.mitre.org/cwss/) and the Common Weakness Risk Analysis Framework (CWRAF) (https://cwe.mitre.org/cwraf/). The mechanisms in CWSS and CWRAF minimize this difficulty by letting organizations model their own business impact considerations into a risk-scoring mechanism.

CWSS provides the mechanism for scoring software's weaknesses in a consistent, flexible, open manner while considering the context and reflecting the weaknesses' impacts against that context. It aims to provide a consistent approach for tools and services prioritizing their static- and dynamic-analysis findings while addressing government, academia, and industry stakeholder needs.

CWRAF uses the core scoring mechanisms from CWSS to let software developers and consumers prioritize their own target list of software weaknesses across their unique portfolio of software applications and projects, focusing on those with the greatest potential to harm their business. To reduce risk, organizations can select appropriate tools and techniques, focus staff training, and define contracting details to ensure outsourced efforts also address the prioritized issues.

CWRAF and CWSS let users create top-n lists for their particular software and business domains, missions, and technology groups. In conjunction with other activities, CWSS and CWRAF help developers and consumers introduce more robust and resilient software into their operational environments.

## Key Discussion Points Between Developers and Consumers, Acquirers, and Project Management

Improving software assurance requires an honest dialog between consumers, acquirers, project managers, and developers on an ongoing basis. Here are some discussion points you can bring up to spark a discussion that will hopefully provide you and them a better understanding of what you and they are doing and need to do to help improve the assurance around your software.

1. Design/Development Practices
    a. Which BSIMM or OpenSAMM activities/practices are followed?
    b. Which SDLC activities are used to directly prevent or mitigate vulnerabilities in the application software? (e.g. threat modeling, automated code analysis (static or dynamic), etc).
    c. Which security controls have been utilized to mitigate specific problems (e.g. authentication, authorization, cryptography)
    d. Which security-related frameworks are used, such as ESAPI or built-in frameworks?

e. Which secure coding rules/practices are followed? (e.g. CERT, MISRA, ISO SC-22, custom). How is conformance enforced (e.g. automated tools during checkin)?

f. What differences, if any, exist between the secure development practices for legacy code, versus newly-developed code?

g. For each implemented IETF RFC, how are the concerns in the RFC's "Security" section mitigated?

h. Which "Top N" vulnerability/attack lists do your development practices actively attempt to address (CWE Top 25, OWASP Top Ten, custom Top-N list)?

2. Third-Party Software Management

   a. Which third-party libraries are used by the software?

   b. How does the development team keep current with third- party libraries so that it does not use code with known vulnerabilities?

   c. How are third-party code changes and vulnerabilities tracked/monitored?

   d. Which third-party libraries were independently examined for vulnerabilities before being included in the software?

3. Detection and Analysis

   a. Which standardized analysis/testing methodologies are used to evaluate the software? (e.g. OWASP ASVS, OSSTMM)

   b. Has an independent 3rd-party review been performed against the software? Did the review cover code implementation, design, architecture, or installation settings?

   d. What tools are used for automated code analysis? Static or dynamic? White box or black box?

   e. Which manual analysis techniques were used?

   f What specifications, data formats, and protocols are used? Were any test case suites or fuzzing tools used to evaluate the implementation (e.g. PROTOS)?

   g. What is the attack surface of the software (in privileged code and overall)?. What metrics are used? Can the attack surface be described in terms of CAPEC?

   h. Which parts of the code have been most recently reviewed?

   i. Which parts of the software contain legacy code whose analysis has been skipped?

4. Compiler/Environment

   a. Which compiler settings are used to reduce or eliminate risk for key weaknesses (e.g. /GS switch)?

   b. Were any compiler warnings ignored when compiling the code? If so, which ones and why?

   c. Was the code compiled using safe libraries?

   d. Which OS features are used to reduce or eliminate the risk of important weaknesses (e.g. DEP, ASLR)?

5. Configuration/Installation

   a. Is the product installed "secure by default"?

   b. Is the product installed so that critical executables, libraries, configuration files, registry keys, etc. cannot be modified by untrusted parties?

   c. Does the software run with limited privileges? If not, how is privilege separation performed?

   d. How does the documentation cover security-relevant settings for administrators to use to lock down the software?

   e. Does the software work under FDCC/USGCB configurations, and/or other secure configurations?

   f. How does the software restrict access to network ports?

6. Vulnerability Response



   a. Is a security response center set up to handle incoming vulnerability reports from external parties?

   b. How easy is it for independent researchers and non-customers to report vulnerabilities?

   c. Are emergency procedures in place to quickly fix issues that are first discovered being exploited in the wild?

   d. Are procedures in place to handle when vulnerabilities are publicly disclosed without notifying the developer or giving sufficient time to produce a patch)?

   e. Is there a sufficiently comprehensive set of information sources that are monitored for reported vulnerabilities in your own software, in third-party products, and competitor/analogous products?

   f. When a new weakness is found by an outside party, how are the software and associated development practices reviewed and modified to ensure that similar weaknesses are also detected and removed?

7. Vulnerability Disclosure

   a. How are consumers of the software notified about new vulnerabilities found in the code?

   b. For vulnerabilities that are publicly disclosed by other parties without a patch, is there a policy to provide public commentary before a patch is available?

   c. Which details are disclosed to customers? What is disclosed to the general public?

   d. Are any credits or compensation provided to independent vulnerability researchers?

8. What kind of evidence or proof can be offered regarding these claims?

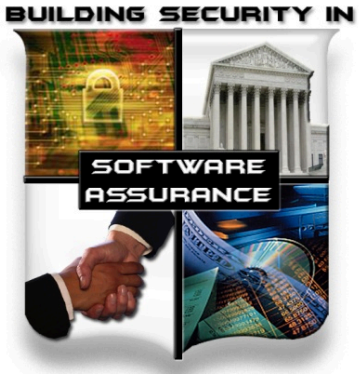
## Using Tools and Other Capabilities to Identify the Top 25

Developers and third-party analysts can use CWE-compatible tools that can map to CWE items in the CWE Top 25. With the advancing maturity and increasing adoption of CWE, most vendors of software analysis tools and services express their findings of weaknesses in code, design, and architecture using CWE identifiers. This common language for expressing weaknesses has eliminated much of the ambiguity and confusion surrounding exactly what the tool or service has found. At the same time, different vendors take different approaches as to how they look for weaknesses and what weaknesses they look for. The CWE Coverage Claims Representation (CCR) is a means for software analysis vendors to convey to their customers exactly which CWE-identified weaknesses they claim to be able to locate in software. The word claim is emphasized since neither the CCR itself nor the CWE Compatibility Program verify or otherwise vet these statements of coverage. The CWE Effectiveness Program will eventually fulfill this role of verification.

The main goal of the CCR is to facilitate the communication of unambiguous statements of the intention of a tool or service to discover specific, CWE-identified weaknesses in software. These statements of claim are intended to allow the providers of software analysis tools and services and the consumers of those tools and services to share a single, unambiguous understanding of the scope of software weakness analysis. CCR wants users of tools and services to be aware and informed of the coverage of the tools and services they make use of in analyzing their software, and when specific classes of weaknesses or individual weaknesses are of specific concern, they can make sure their tools and services are at least trying to find them. Having a mis-match between an organization's focus and the capabilities of their tools and services is not something to be discovered after using and depending on them, but rather is something that should be addressed in the initial discussions and exploration of bringing those capabilities to bear for the organization.

It is anticipated that the CCR will also foster innovation in the technology of software analysis tools and services by allowing vendors to clearly state their intentions with respect to weakness discovery and understand more clearly when there is a need for targeting additional weaknesses to address their customer's concerns. Currently, a tool that does a very deep analysis on a small subset of the entire set of CWE-defined weaknesses may be judged as inadequate by potential customers since, by definition, it fails to discover a broad set of weaknesses. However, with the CCR, the tool provider could supply a CCR document for that tool, clearly setting expectations as to the set of weaknesses that the tool attempts to discover. Tool consumers could then evaluate tools based on what specific CWE-identified weaknesses those tools claim to discover and how that coverage fits within their needs, rather than comparing it to the entire set of CWE-defined weaknesses.

# End – Part 1


# Measuring Software Security

## Moving Towards Software Assurance Automation

## Part 2

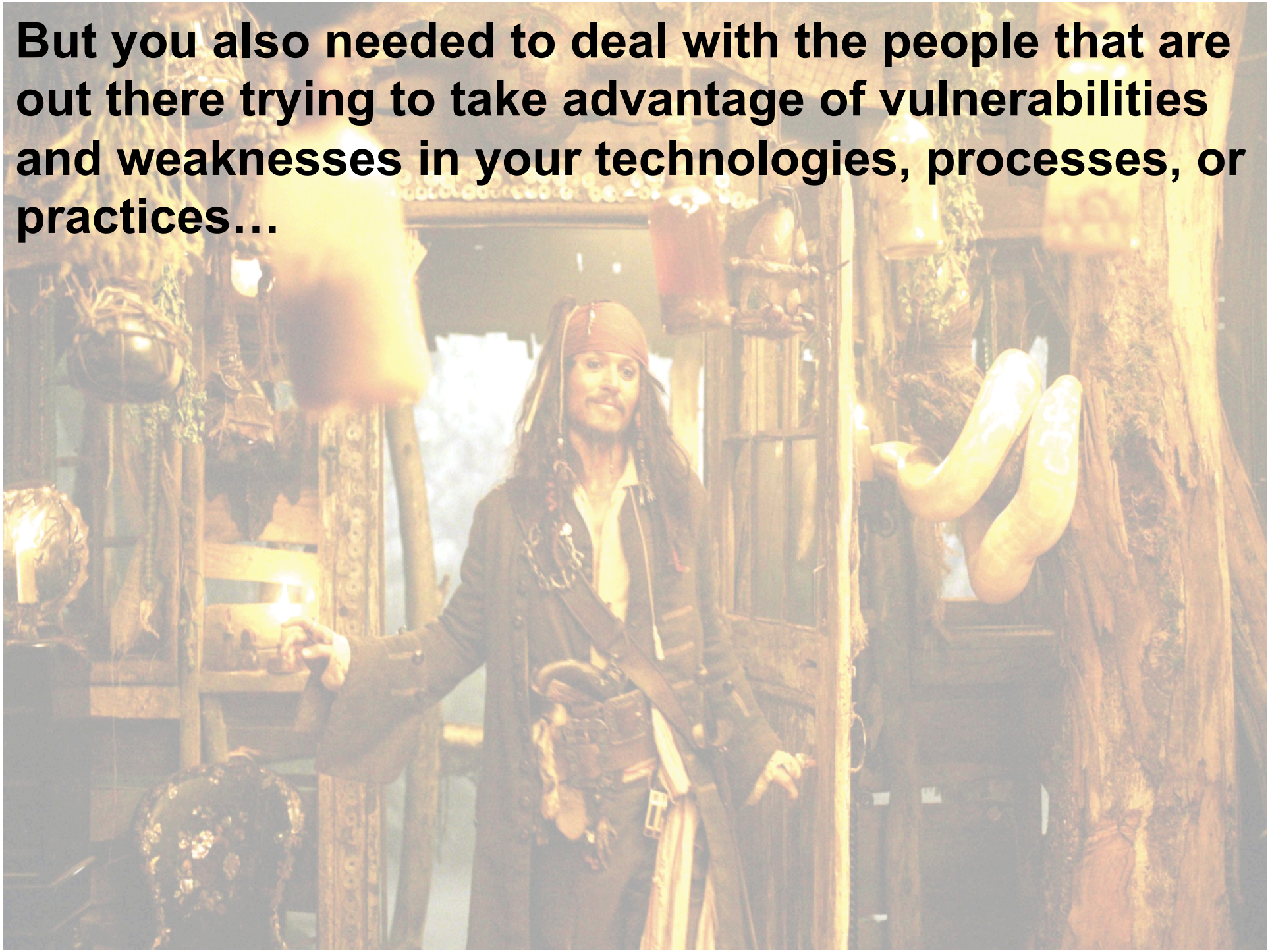
**Joe Jarzombek, DHS Director of SwA**
**Robert Martin, MITRE CWE Project Lead**

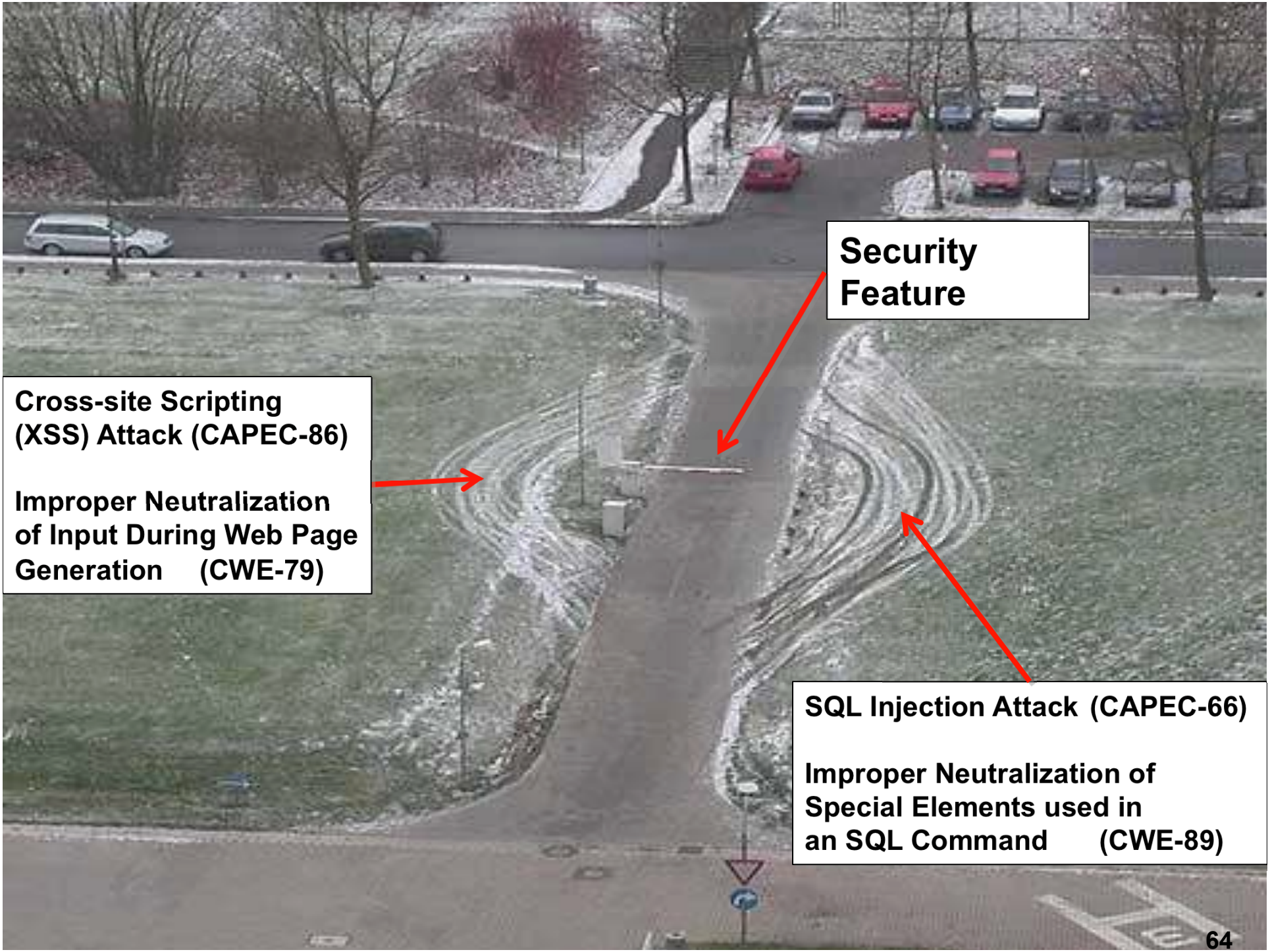
**August 21, 2012**


Homeland Security

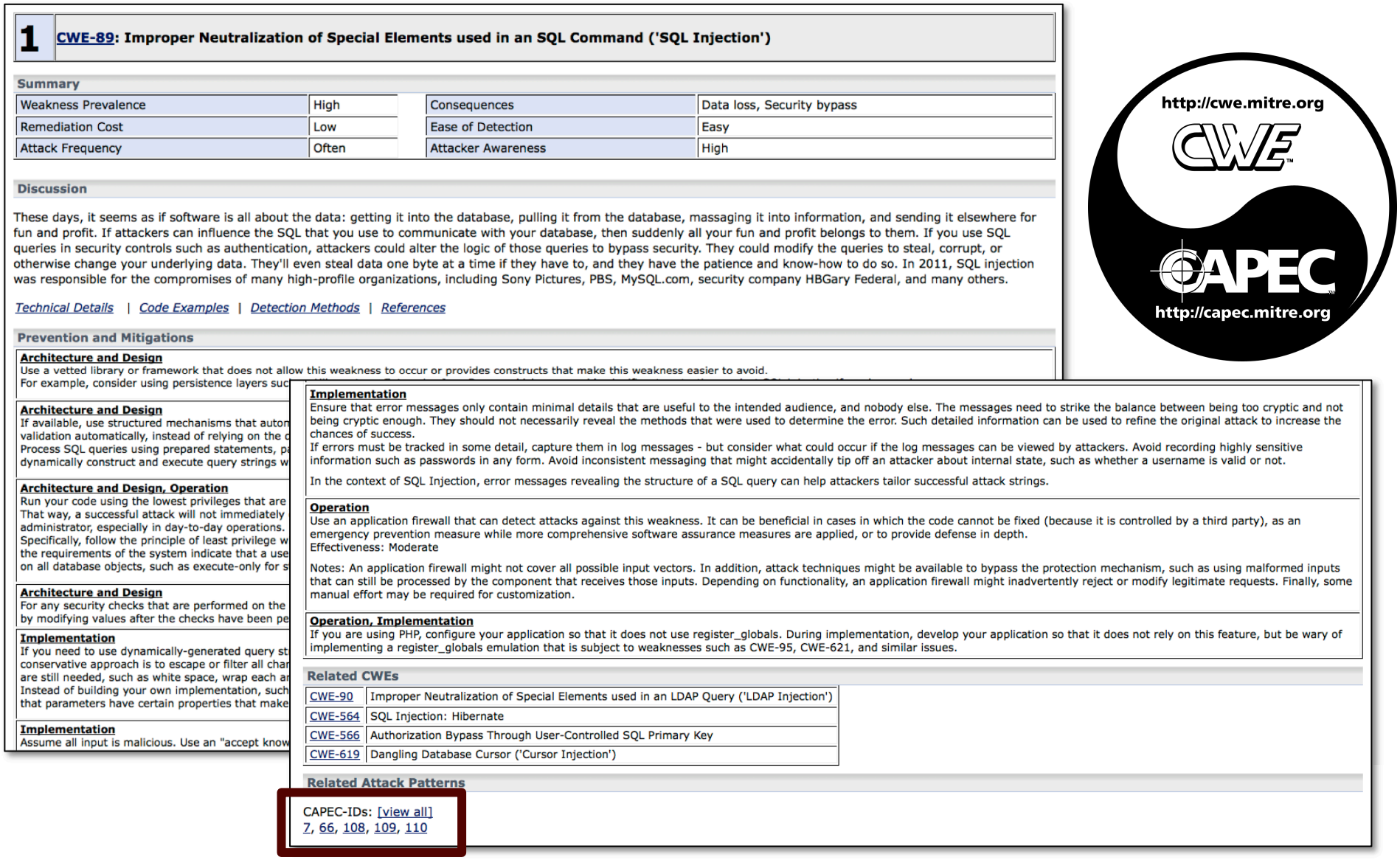
MITRE

**But you also needed to deal with the people that are out there trying to take advantage of vulnerabilities and weaknesses in your technologies, processes, or practices…**




Security Feature
Cross-site Scripting (XSS) Attack (CAPEC-86)
Improper Neutralization of Input During Web Page Generation (CWE-79)
SQL Injection Attack (CAPEC-66)
Improper Neutralization of Special Elements used in an SQL Command (CWE-89)

# What are the Attacks that would be Effective Against Your Weaknesses?



**1** CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

**Summary**

| | | | |
|---|---|---|---|
| Weakness Prevalence | High | Consequences | Data loss, Security bypass |
| Remediation Cost | Low | Ease of Detection | Easy |
| Attack Frequency | Often | Attacker Awareness | High |

**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. In 2011, SQL injection was responsible for the compromises of many high-profile organizations, including Sony Pictures, PBS, MySQL.com, security company HBGary Federal, and many others.

*Technical Details* | *Code Examples* | *Detection Methods* | *References*

**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 suc

**Architecture and Design**
If available, use structured mechanisms that auton
validation automatically, instead of relying on the c
Process SQL queries using prepared statements, pa
dynamically construct and execute query strings w

**Architecture and Design, Operation**
Run your code using the lowest privileges that are
That way, a successful attack will not immediately
administrator, especially in day-to-day operations.
Specifically, follow the principle of least privilege w
the requirements of the system indicate that a use
on all database objects, such as execute-only for s

**Architecture and Design**
For any security checks that are performed on the
by modifying values after the checks have been pe

**Implementation**
If you need to use dynamically-generated query st
conservative approach is to escape or filter all char
are still needed, such as white space, wrap each ar
Instead of building your own implementation, such
that parameters have certain properties that make

**Implementation**
Assume all input is malicious. Use an "accept know

**Implementation**
Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success.
If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

In the context of SQL Injection, error messages revealing the structure of a SQL query can help attackers tailor successful attack strings.

**Operation**
Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.
Effectiveness: Moderate

Notes: An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

**Operation, Implementation**
If you are using PHP, configure your application so that it does not use register_globals. During implementation, develop your application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

**Related CWEs**

| | |
|---|---|
| CWE-90 | Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection') |
| CWE-564 | SQL Injection: Hibernate |
| CWE-566 | Authorization Bypass Through User-Controlled SQL Primary Key |
| CWE-619 | Dangling Database Cursor ('Cursor Injection') |

**Related Attack Patterns**

CAPEC-IDs: [view all]
7, 66, 108, 109, 110

http://cwe.mitre.org
CWE
CAPEC
http://capec.mitre.org



MITRE

# Software, Network Traffic, Physical, Social Engineering, and Supply Chain Attack Patterns

CAPEC Common Attack Pattern Enumeration and Classification
A Community Knowledge Resource for Building Secure Software

Home > CAPEC List > CAPEC-1000: Mechanism of Attack (Release 1.7.1) Search by ID: Go

CAPEC List
Full CAPEC Dictionary
Methods of Attack View
Reports
About CAPEC
Documents
Resources
Community
Related Activities
Collaboration List
T-Shirt
News & Events
Calendar
Free Newsletter
Compatibility
Program Requirements
Make a Declaration
Contact Us
Search the Site

## CAPEC-1000: Mechanism of Attack

Definition Graph List Slice XML.zip

### Mechanism of Attack

**View ID:** 1000 *(View: Graph)* **Status:** Draft

### View Data

**View Structure:** Graph

**View Objective**

### Relationships

| Nature | Type | ID | Name | Description | V |
|---|---|---|---|---|---|
| HasMember | | 118 | Data Leakage Attacks | | 1000 |
| HasMember | | 119 | Resource Depletion | | 1000 |
| HasMember | | 152 | Injection (Injecting Control Plane content through the Data Plane) | | 1000 |
| HasMember | | 156 | Spoofing | | 1000 |
| HasMember | | 172 | Time and State Attacks | | 1000 |
| HasMember | | 210 | Abuse of Functionality | | 1000 |
| HasMember | | 223 | Probabilistic Techniques | | 1000 |
| HasMember | | 225 | Exploitation of Authentication | | 1000 |
| HasMember | | 232 | Exploitation of Privilege/Trust | | 1000 |
| HasMember | | 255 | Data Structure Attacks | | 1000 |
| HasMember | | 262 | Resource Manipulation | | 1000 |
| HasMember | A | 286 | Network Reconnaissance | | 1000 |
| HasMember | A | 403 | Social Engineering Attacks | | 1000 |
| HasMember | | 436 | Physical Security Attacks | | 1000 |
| HasMember | A | 437 | Supply Chain Attacks | | 1000 |

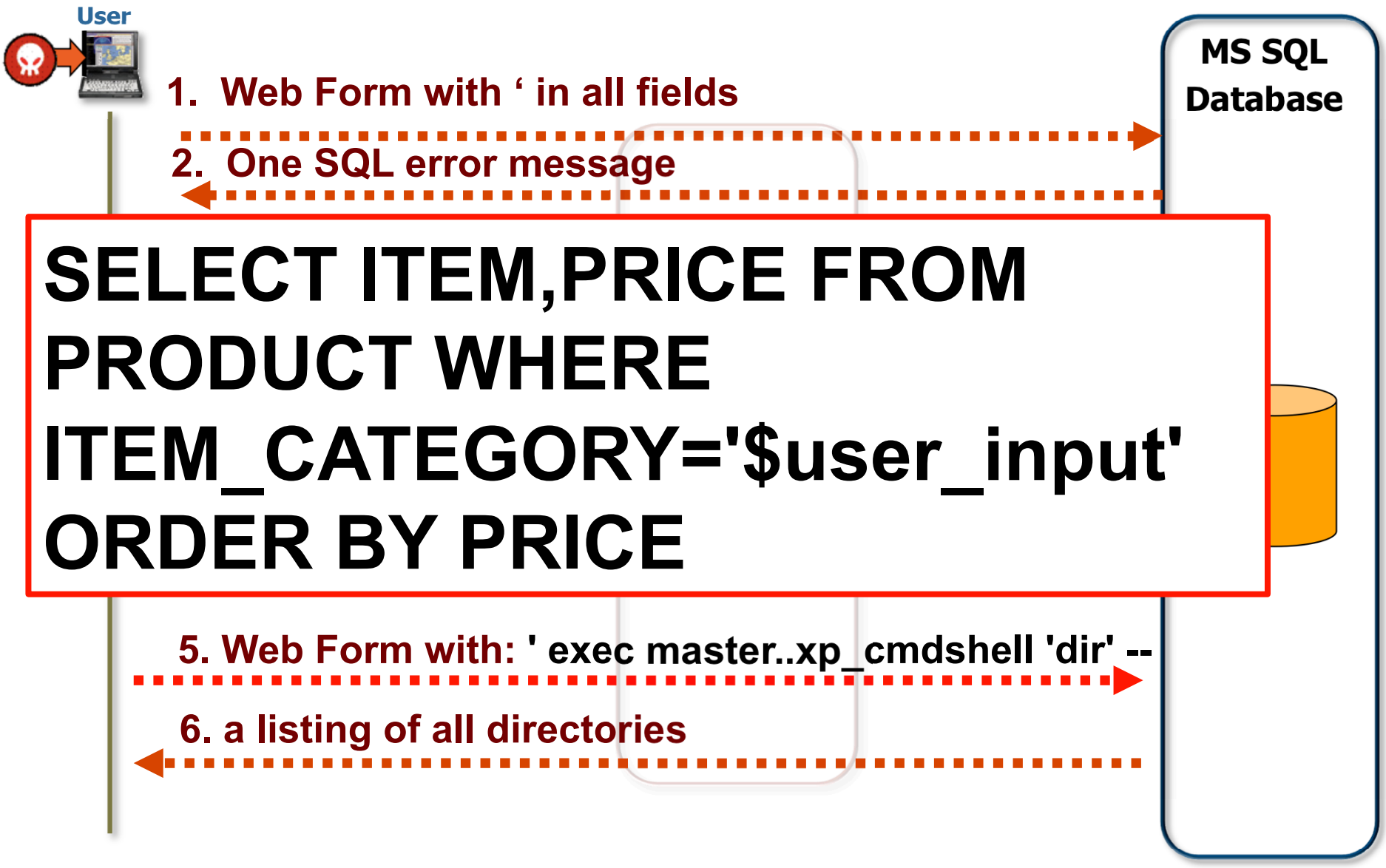
| | CAPECs in this view | | Total CAPECs |
|---|---|---|---|
| **Total** | **412** | out of | 474 |
| **Views** | 0 | out of | 6 |
| **Categories** | 19 | out of | 68 |
| **Attack Patterns** | 400 | out of | 400 |

BACK TO TOP

**Page Last Updated:** May 04, 2012

MITRE

# SQL Injection Attack Execution Flow



# Simple test case for SQL Injection

**Test Case 1: Single quote SQL injection of registration page web form fields**

**Test Case Goal:** Ensure SQL syntax single quote character entered in registration page web form fields does not cause abnormal SQL behavior

**Context:**

- This test case is part of a broader SQL injection syntax exploration suite of tests to probe various potential injection points for susceptibility to SQL injection. If this test case fails, it should be followed-up with test cases from the SQL injection experimentation test suite.

**Preconditions:**

- Access to system registration page exists
- Registration page web form field content are used by system in SQL queries of the system database upon page submission
- User has the ability to enter free-form text into registration page web form fields

**Test Data:**

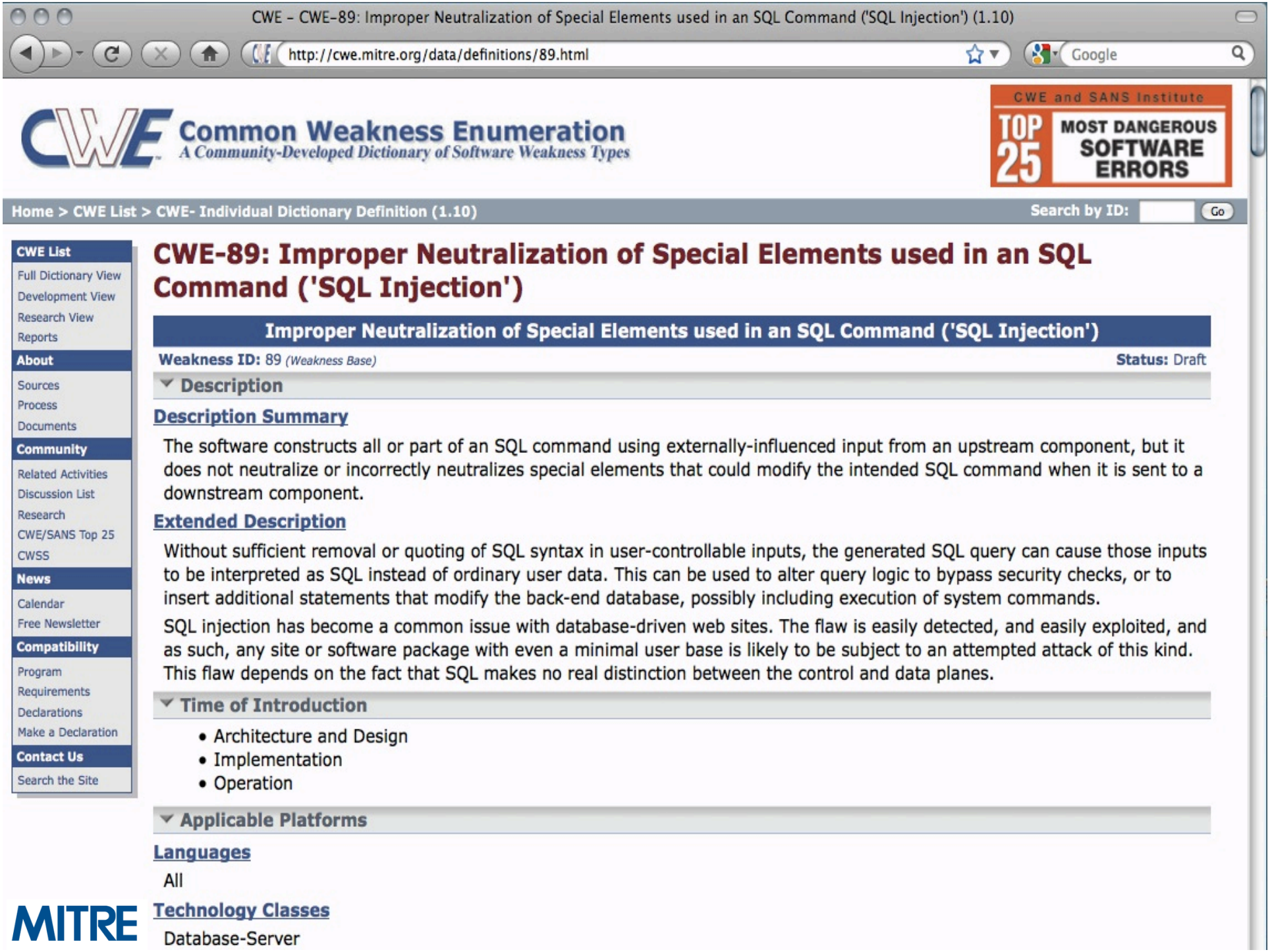
- ASCII single quote character

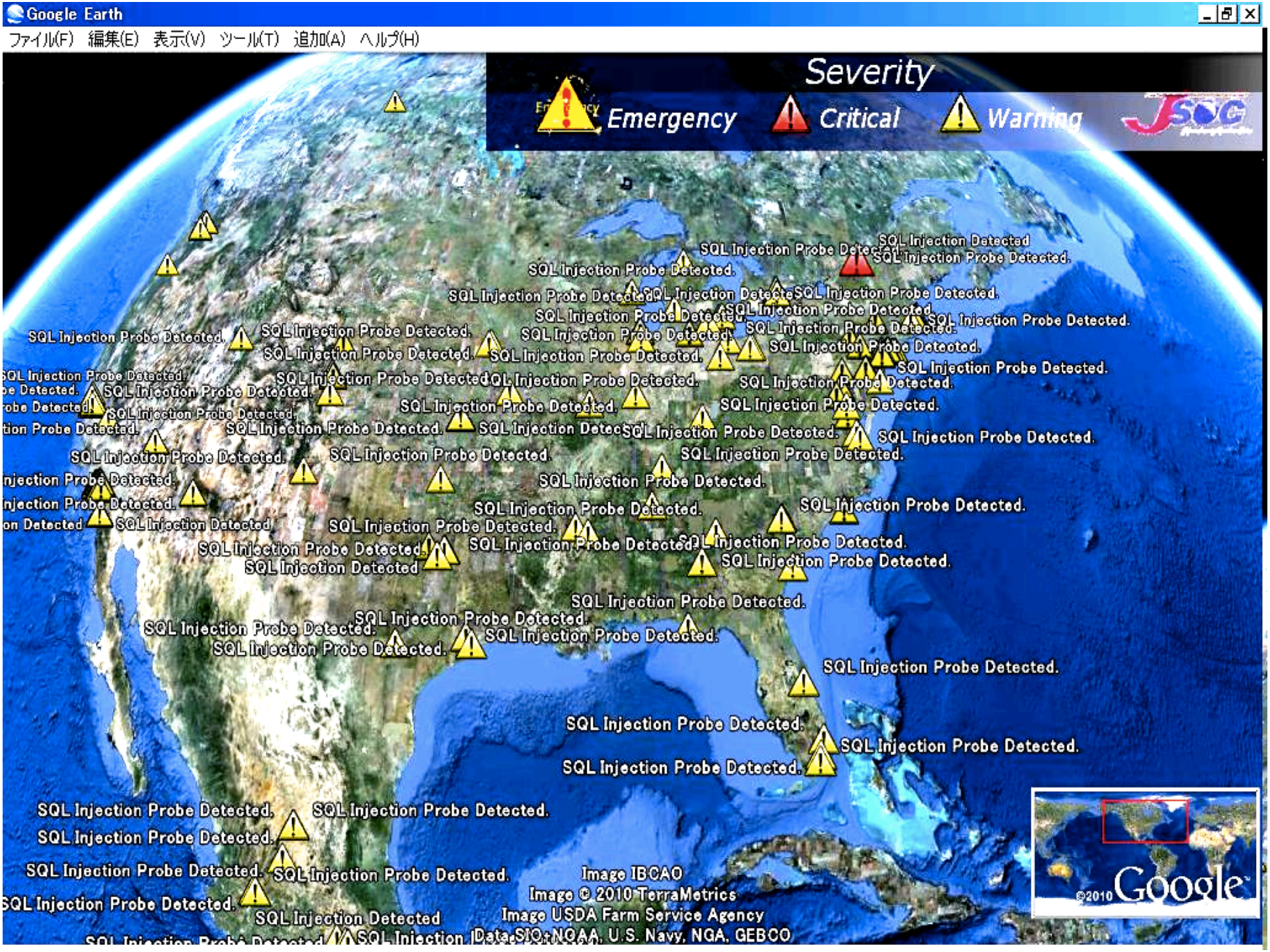
**Action Steps:**

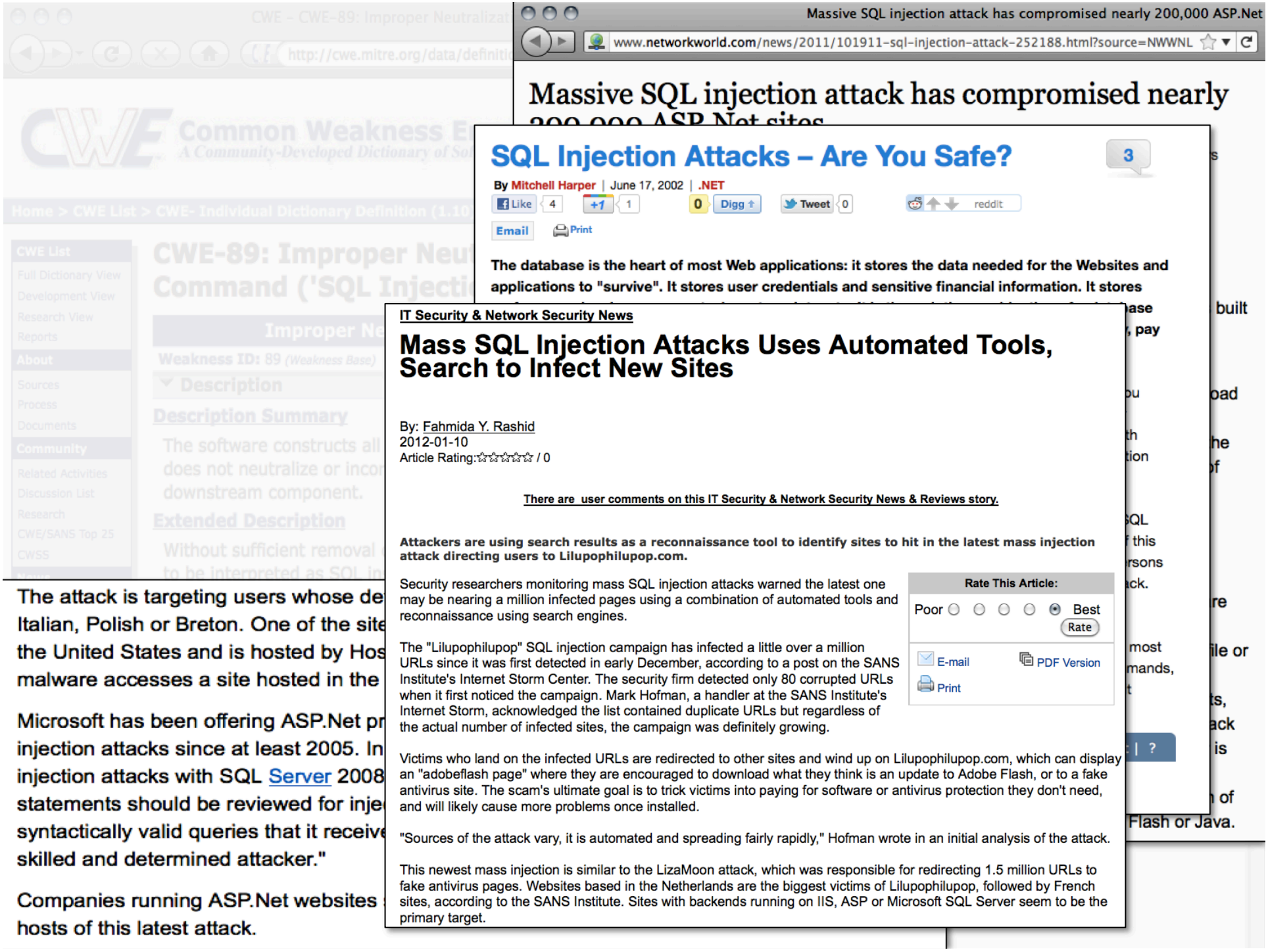
- Enter single quote character into each web form field on the registration page
- Submit the contents of the registration page

**Postconditions:**

- Test case fails if SQL error is thrown
- Test case passes if page submission succeeds without any SQL errors


CWE – CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (1.10)
http://cwe.mitre.org/data/definitions/89.html
Google
Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types
CWE and SANS Institute
TOP 25 MOST DANGEROUS SOFTWARE ERRORS
Home > CWE List > CWE- Individual Dictionary Definition (1.10)
Search by ID:
Go
CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
Community
Related Activities
Discussion List
Research
CWE/SANS Top 25
CWSS
News
Calendar
Free Newsletter
Compatibility
Program
Requirements
Declarations
Make a Declaration
Contact Us
Search the Site
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
Weakness ID: 89 (Weakness Base)
Status: Draft
Description
Description Summary
The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.
Extended Description
Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.
SQL injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.
Time of Introduction
• Architecture and Design
• Implementation
• Operation
Applicable Platforms
Languages
All
Technology Classes
Database-Server
MITRE


Google Earth
ファイル(F) 編集(E) 表示(V) ツール(T) 追加(A) ヘルプ(H)
Severity
Emergency
Critical
Warning
JSOC
SQL Injection Probe Detected.
SQL Injection Detected
Image IBCAO
Image © 2010 TerraMetrics
Image USDA Farm Service Agency
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
©2010 Google



Massive SQL injection attack has compromised nearly 200,000 ASP.Net

www.networkworld.com/news/2011/101911-sql-injection-attack-252188.html?source=NWWNL

Massive SQL injection attack has compromised nearly

SQL Injection Attacks – Are You Safe?

3

By Mitchell Harper | June 17, 2002 | .NET

Like 4 +1 1 0 Digg Tweet 0 reddit

Email Print

**The database is the heart of most Web applications: it stores the data needed for the Websites and applications to "survive". It stores user credentials and sensitive financial information. It stores**

IT Security & Network Security News

## Mass SQL Injection Attacks Uses Automated Tools, Search to Infect New Sites

By: Fahmida Y. Rashid
2012-01-10
Article Rating:☆☆☆☆☆ / 0

There are user comments on this IT Security & Network Security News & Reviews story.

**Attackers are using search results as a reconnaissance tool to identify sites to hit in the latest mass injection attack directing users to Lilupophilupop.com.**

Security researchers monitoring mass SQL injection attacks warned the latest one may be nearing a million infected pages using a combination of automated tools and reconnaissance using search engines.

Rate This Article:
Poor ○ ○ ○ ○ ○ Best
Rate

E-mail PDF Version
Print

The "Lilupophilupop" SQL injection campaign has infected a little over a million URLs since it was first detected in early December, according to a post on the SANS Institute's Internet Storm Center. The security firm detected only 80 corrupted URLs when it first noticed the campaign. Mark Hofman, a handler at the SANS Institute's Internet Storm, acknowledged the list contained duplicate URLs but regardless of the actual number of infected sites, the campaign was definitely growing.

Victims who land on the infected URLs are redirected to other sites and wind up on Lilupophilupop.com, which can display an "adobeflash page" where they are encouraged to download what they think is an update to Adobe Flash, or to a fake antivirus site. The scam's ultimate goal is to trick victims into paying for software or antivirus protection they don't need, and will likely cause more problems once installed.

"Sources of the attack vary, it is automated and spreading fairly rapidly," Hofman wrote in an initial analysis of the attack.

This newest mass injection is similar to the LizaMoon attack, which was responsible for redirecting 1.5 million URLs to fake antivirus pages. Websites based in the Netherlands are the biggest victims of Lilupophilupop, followed by French sites, according to the SANS Institute. Sites with backends running on IIS, ASP or Microsoft SQL Server seem to be the primary target.

The attack is targeting users whose de
Italian, Polish or Breton. One of the site
the United States and is hosted by Hos
malware accesses a site hosted in the

Microsoft has been offering ASP.Net pr
injection attacks since at least 2005. In
injection attacks with SQL Server 2008
statements should be reviewed for inje
syntactically valid queries that it receive
skilled and determined attacker."

Companies running ASP.Net websites
hosts of this latest attack.

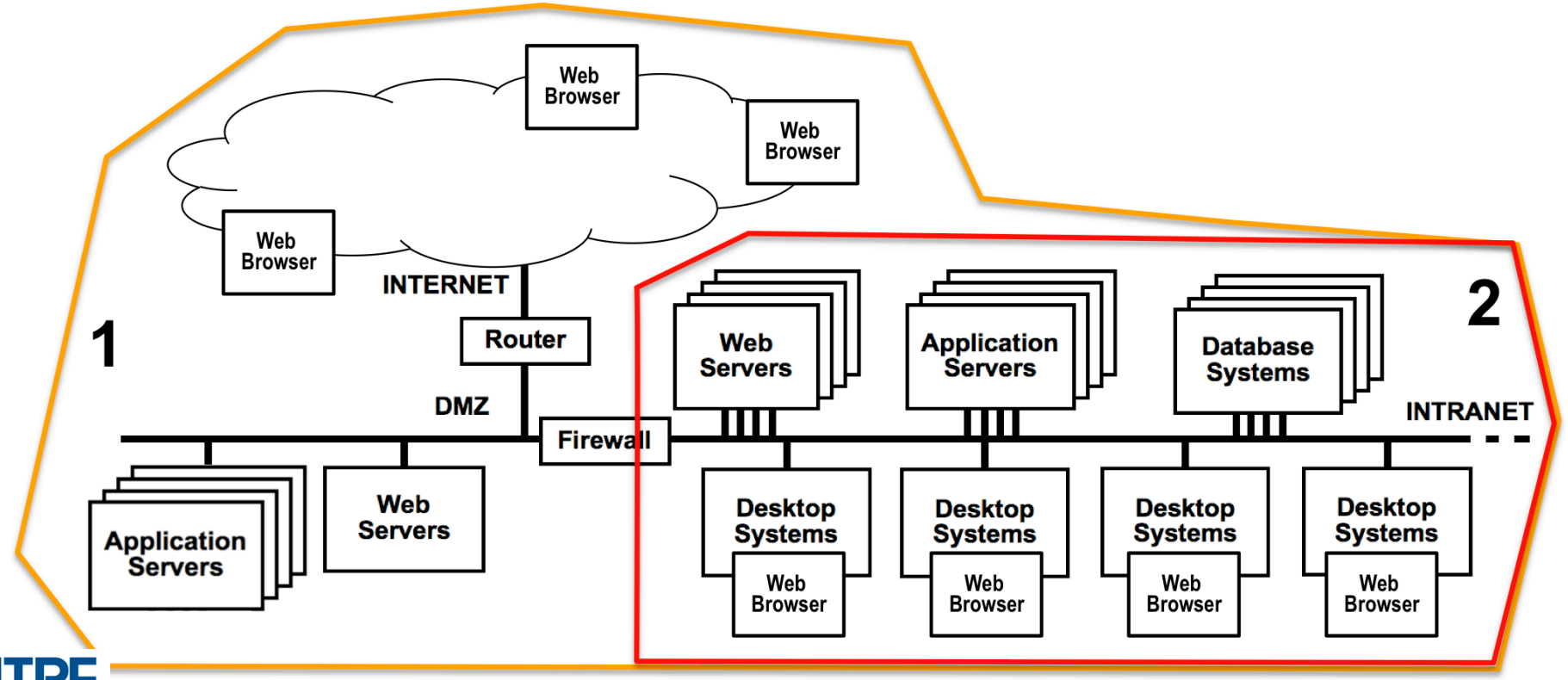
# Scoring Weaknesses Based on Context

**Archetypes:**

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

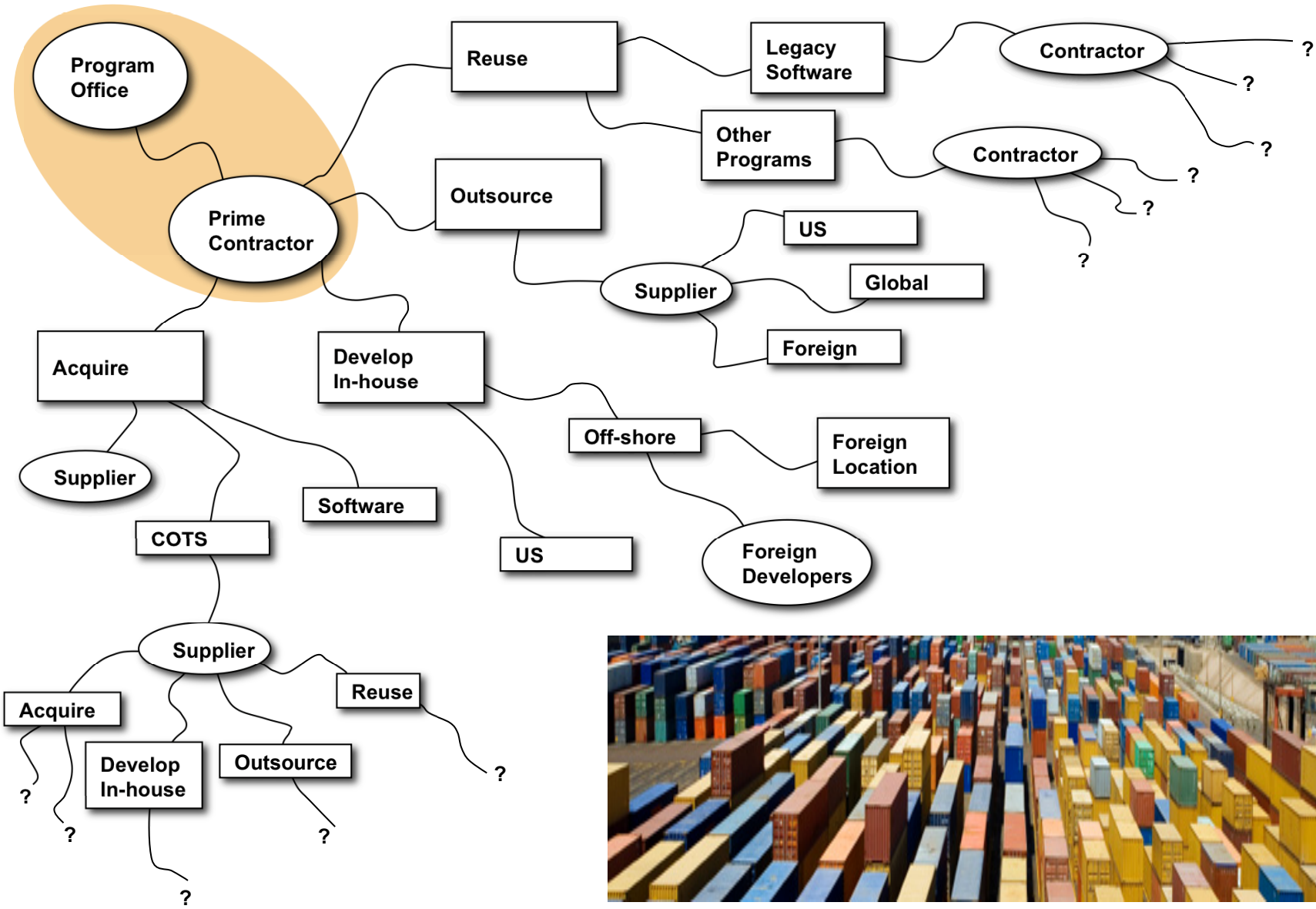
**Vignettes:**

1. Web-based Retail Provider
2. Intranet resident health records management system of hospital



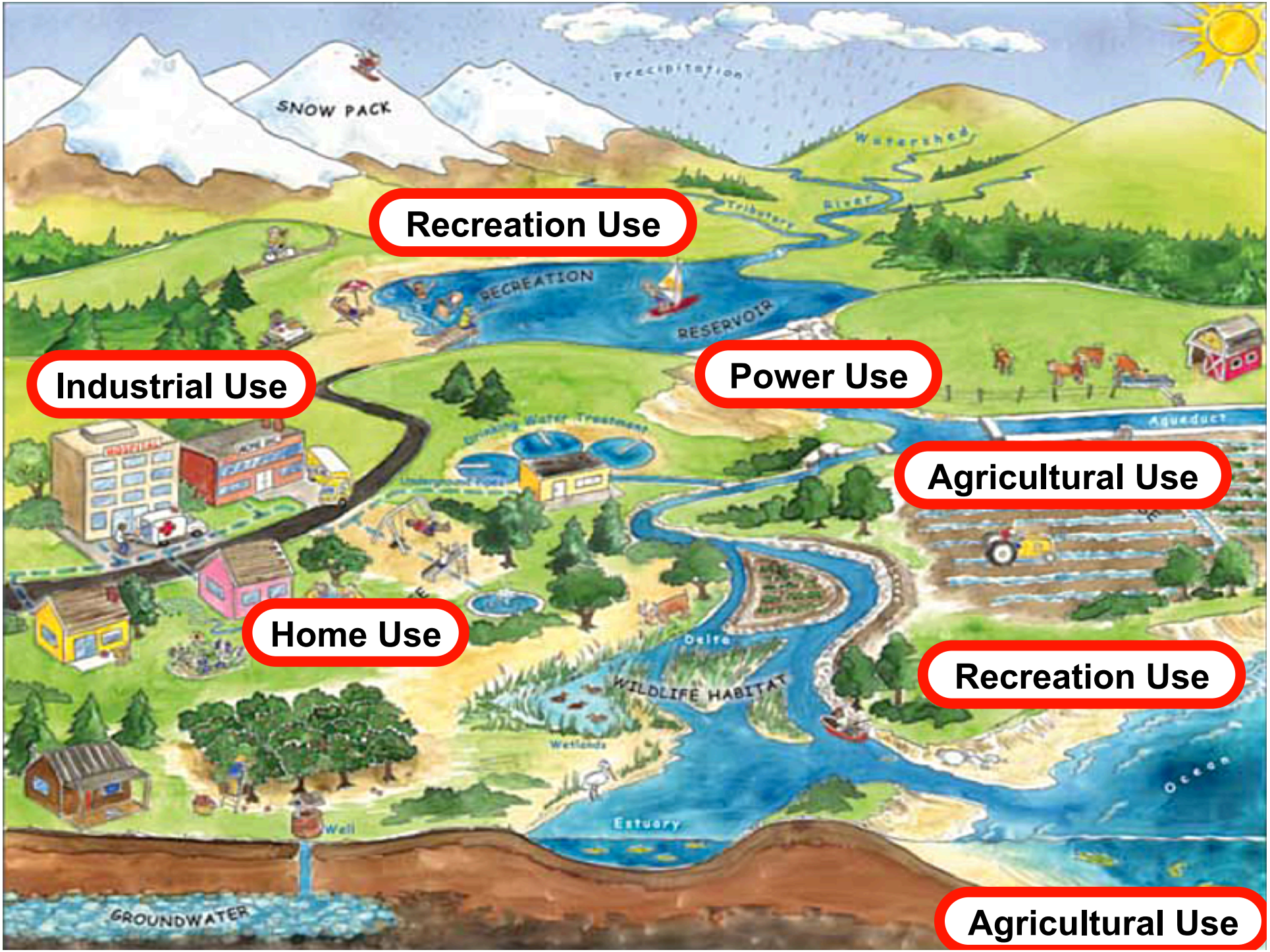

MITRE

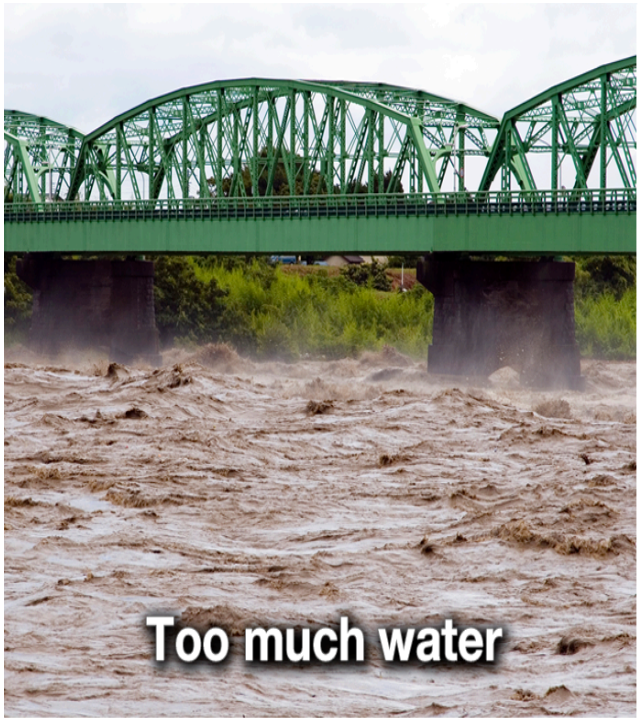
# The Software Supply Chain






* "Scope of Supplier Expansion and Foreign Involvement" graphic in DACS www.softwaretechnews.com Secure Software Engineering, July 2005 article "Software Development Security: A Risk Management Perspective" synopsis of May 2004 GAO-04-678 report "Defense Acquisition: Knowledge of Software Suppliers Needed to Manage Risks"


Recreation Use
Industrial Use
Power Use
Agricultural Use
Home Use
Recreation Use
Agricultural Use
SNOW PACK
Precipitation
Watershed
Tributary
River
RECREATION
RESERVOIR
Drinking Water Treatment
Aqueduct
Delta
WILDLIFE HABITAT
Wetlands
Estuary
Ocean
Well
GROUNDWATER


Too much water




# The Software Industry's "Clean Water Act" Alternative

**Robert A. Martin and Steven M. Christey** | MITRE

**Following the water industry's example, the authors advocate for implementing processes that can examine software and remove the most dangerous contaminants, given its intended use.**

Much like the water we use in diverse daily activities across all aspects of our world's ecosystem, the actual sources of, and manner in which we receive, the software in our cyberecosystem are often unknown and possibly unknowable. Over time, the water industry has developed water-quality measurements and methods that give users trust in the fact that harmful water qualities aren't present. This is due to the industry's technical ability to specify and measure water qualities, such as temperature, hardness, volume, and pollutants, as well as the regulatory framework that mandates that those who offer water check these characteristics for dangerous levels according to the water's intended use. When harmful levels are detected, mitigations and controls can be applied and verified, including water softeners, filtration, settling ponds, and cooling towers.

The software industry must implement similar processes and technical methods to examine software for dangerous contaminants, given its intended use, and ensure that appropriate mitigations and controls are in place to remove the harmful characteristics. Several software assurance strategic initiatives, cosponsored by the US Department of Homeland Security National Cyber Security Division, attempt to make this process easier. The Common Weakness Enumeration (CWE; https://cwe.mitre.org) offers the industry a list of potentially dangerous software contaminants, and the Common Weakness Scoring System (CWSS; https://cwe.mitre.org/cwss) and the Common Weakness Risk Analysis Framework (CWRAF; https://cwe.mitre.org/cwraf) provide a standard method to identify which of these contaminants are most harmful to a particular organization, given the software's intended use.

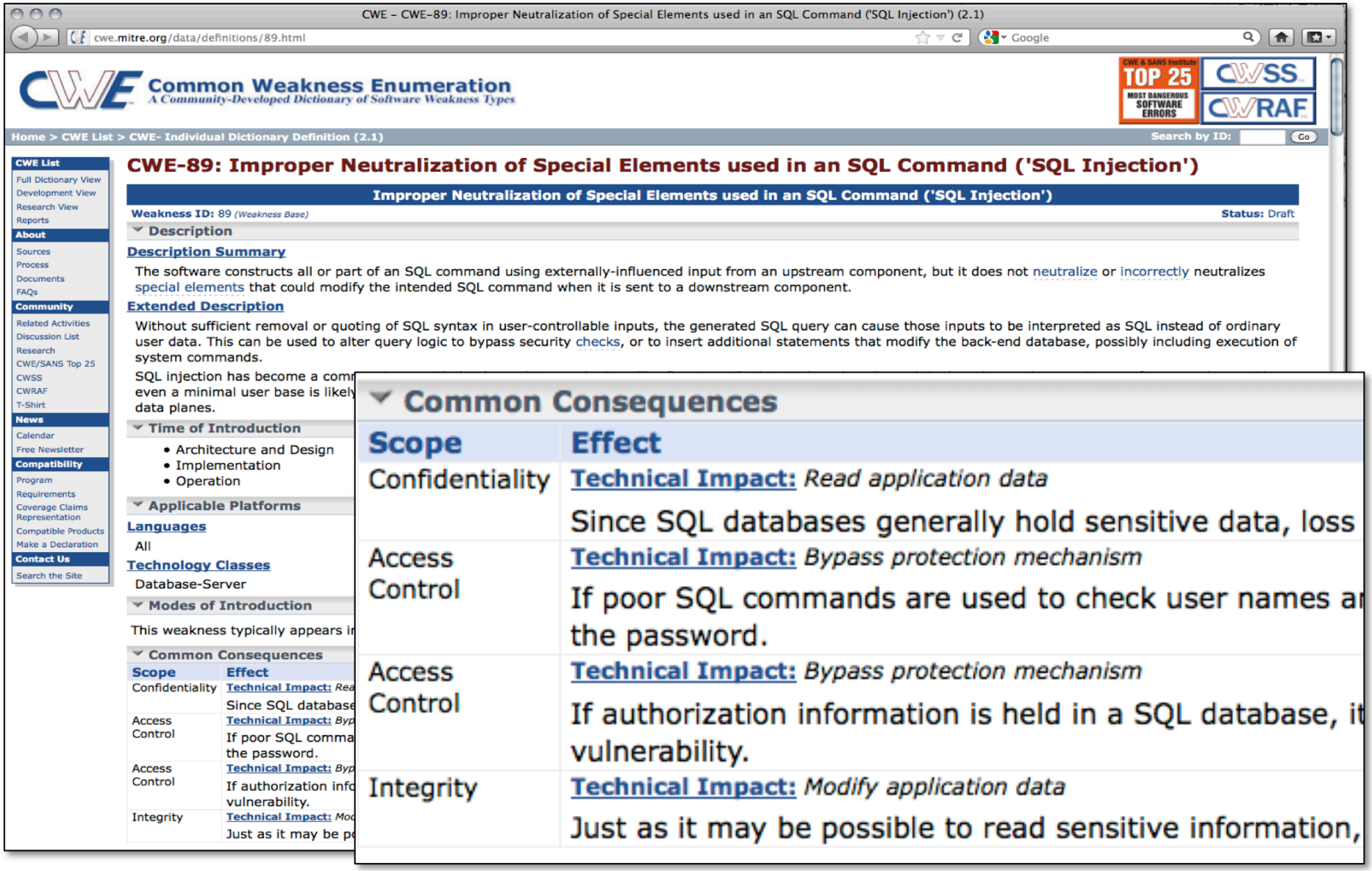
In this article, we define an approach for organizations to document software's security-relevant capabilities and rank the various potential technical impacts from CWEs so those CWEs with the most impact to an organization can be prioritized for mitigation. By addressing vulnerable software and finding systematic and verifiable ways to remove these weaknesses, software providers can improve customers' trust in their systems and possibly avoid a regulatory solution, which might have unintended consequences.

## Background

In the late 1990s, the sharing, discussing, measuring, and reporting of activities surrounding software products' vulnerabilities was unorganized and cumbersome to manage. With the help of industry, academia, and government, MITRE attempted to change this by introducing the Common Vulnerabilities and Exposures (CVE) effort (https://cve.mitre.org). CVE lets

 Copublished by the IEEE Computer and Reliability Societies 

# Technical Impacts – Common Consequences



## Technical Impacts – Common Weakness Risk Analysis Framework (CWRAF)

1. Modify data
2. Read data
3. DoS: unreliable execution
4. DoS: resource consumption
5. Execute unauthorized code or commands
6. Gain privileges / assume identity
7. Bypass protection mechanism
8. Hide activities

## Technical Impacts for CWE Entries

Note that this list is likely to change in future CWE versions.

CWE-89 (SQL Injection) has three technical impacts as listed in the Common_Consequences element of the CWE entry:

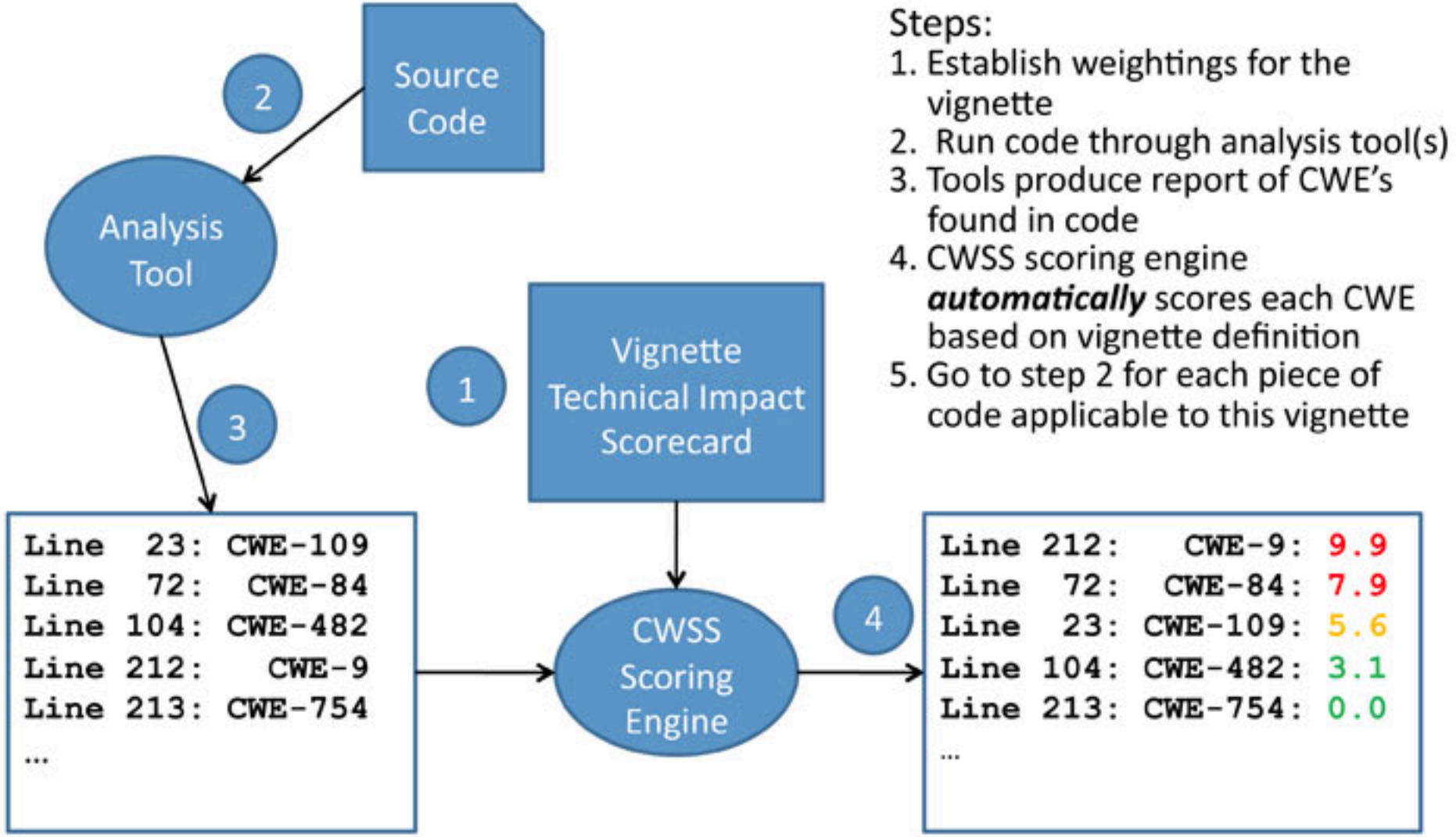
- Read application data
- Modify application data
- Bypass protection mechanism

For CWE-120 (Classic Buffer Overflow), the listed technical impacts are:

- Execute unauthorized code or commands
- DoS: crash / exit / restart

| ID | Name | Max Subscore | Technical Impacts and Importance Subscores |
|---|---|---|---|
| CWE-89 | SQL Injection | 8 | * Read data (8)<br>* Modify data (8)<br>* Bypass protection mechanism (7) |
| CWE-120 | Classic Buffer Overflow | 10 | * Execute unauthorized code or commands (10)<br>* DoS: unreliable execution (4) |

# Scoring Weaknesses Discovered in Code using CWSS



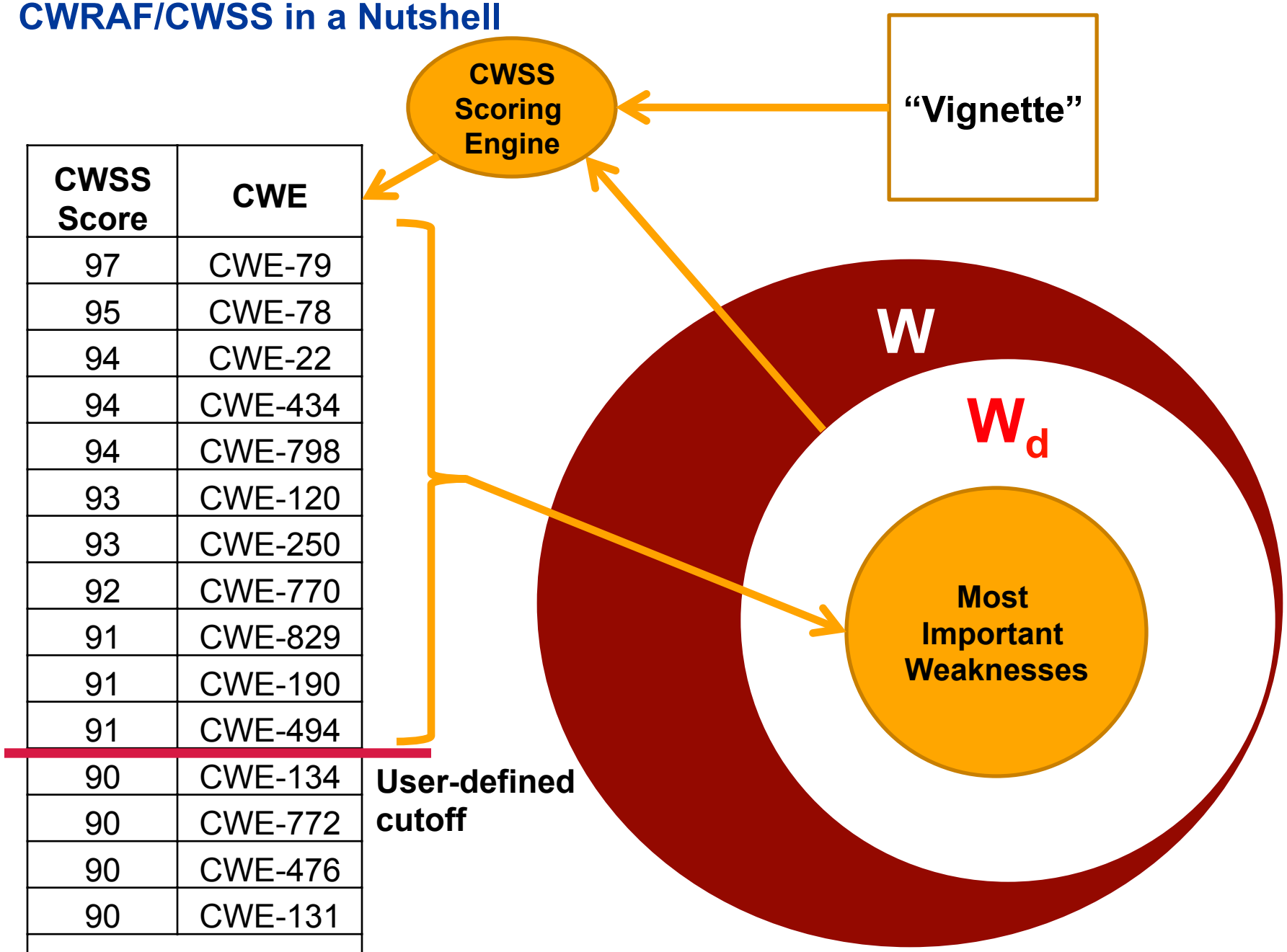

Steps:

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

MITRE

*Step 1 is only done once – the rest is automatic*

# CWRAF/CWSS in a Nutshell

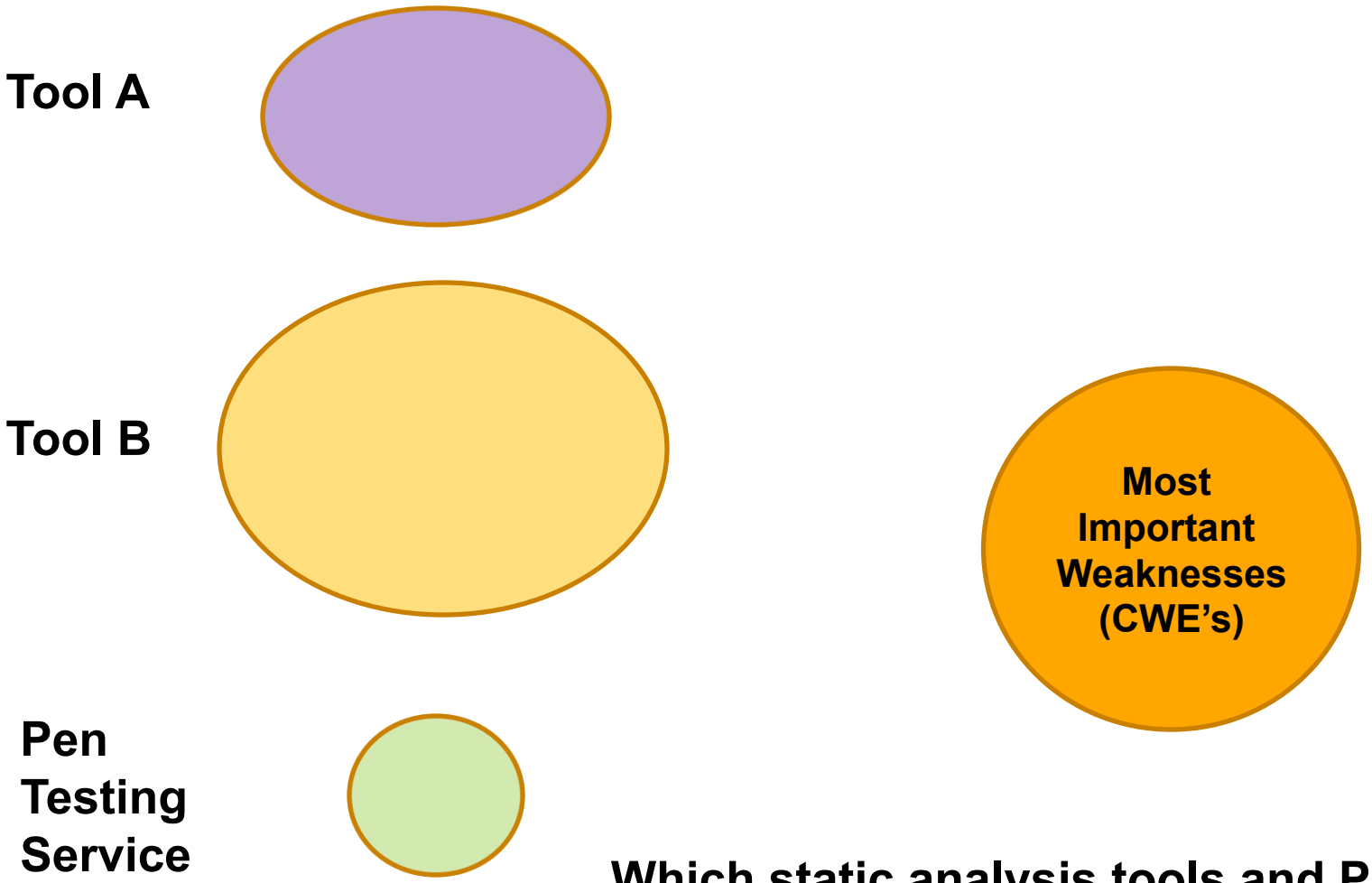



| CWSS Score | CWE |
|---|---|
| 97 | CWE-79 |
| 95 | CWE-78 |
| 94 | CWE-22 |
| 94 | CWE-434 |
| 94 | CWE-798 |
| 93 | CWE-120 |
| 93 | CWE-250 |
| 92 | CWE-770 |
| 91 | CWE-829 |
| 91 | CWE-190 |
| 91 | CWE-494 |
| 90 | CWE-134 |
| 90 | CWE-772 |
| 90 | CWE-476 |
| 90 | CWE-131 |
| … | |

**W is all possible weaknesses; Wd is all known weaknesses (CWE)**

# CWE Coverage Claims

**Set of CWE's a capability *claims* to cover**




**Which static analysis tools and Pen Testing services find the CWE's I care about?**

MITRE

# 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** | **Top 10 List** |

**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**

MITRE

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
FAQs
Community
SwA On-Ramp
T-Shirt
Discussion List
Discussion Archives
Scoring
CWSS
CWRAF
CWE/SANS Top 25
Compatibility
Requirements
Coverage Claims Representation
Compatible Products
Make a Declaration
News
Calendar
Free Newsletter
Contact Us
Search the Site

# Common Weakness Risk Analysis Framework (CWRAF™)

CWRAF provides a framework for scoring software weaknesses in a consistent, flexible, open manner, while accommodating context for the various business domains. It is a collaborative, community-based effort that is addressing the needs of its stakeholders across government, academia, and industry. CWRAF is a part of the Common Weakness Enumeration (CWE™) project, co-sponsored by the Software Assurance program in the National Cyber Security Division (NCSD) of the US Department of Homeland Security (DHS).

### CWRAF benefits:

- Includes a mechanism for measuring risk of security errors ("weaknesses") in a way that is closely linked with the risk to an organization's business or mission.
- Supports the automatic selection and prioritization of relevant weaknesses, customized to the specific needs of the organization's business or mission.
- Can be used by organizations in conjunction with the Common Weakness Scoring System (CWSS™) to identify the most important weaknesses for their business domains, in order to inform their acquisition and protection activities as one part of the larger process of achieving software assurance.

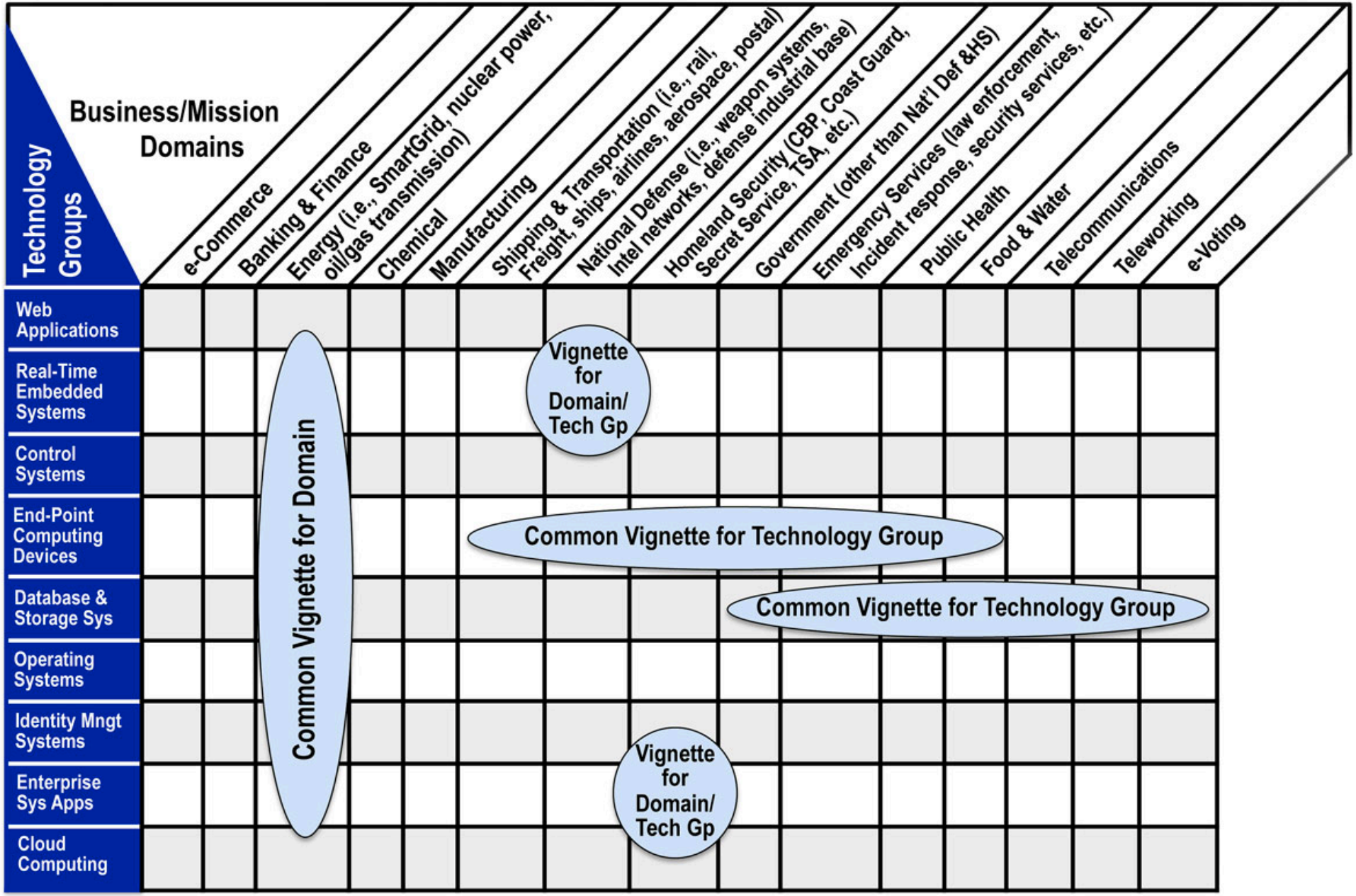
CWRAF and CWSS allow users to rank classes of weaknesses independent of any particular software package, in order to prioritize them relative to each other (e.g., "buffer overflows are higher priority than memory leaks"). This approach, sometimes referred to as a "Top-N list," is used by the CWE/SANS Top 25, OWASP Top Ten, and similar efforts. CWRAF and CWSS allow users to create their own custom Top-N lists.

## CWRAF Version 0.8.1

- Introduction
  - How to Use CWRAF
  - Relationships between CWRAF, CWSS, and CWE
- CWSS Scoring in CWRAF
  - Scoring Weakness Findings Using Vignettes

Section Contents
CWRAF
Introduction
CWSS Scoring in CWRAF
Creating Your Own Vignettes
Future Versions and Activities
Change Log
Vignettes
Tech Groups and Domains
Archetypes
CWRAF FAQs
Other Items of Interest
CWSS
Terms of Use

# Vignettes – Technology Groups & Business/Mission Domains




Common Weakness Risk Assessment Framework uses Vignettes with Archetypes to identify top CWEs in respective Domain/Technology Groups

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
FAQs
Community
SwA On-Ramp
T-Shirt
Discussion List
Discussion Archives
Scoring
CWSS
CWRAF
CWE/SANS Top 25
Compatibility
Requirements
Coverage Claims Representation
Compatible Products
Make a Declaration
News
Calendar
Free Newsletter
Contact Us
Search the Site

# CWRAF - Vignette Overview Matrix



http://cwe.mitre.org/cwraf/

Section Contents
CWRAF
Introduction
CWSS Scoring in CWRAF
Creating Your Own Vignettes
Future Versions and Activities
Change Log
CWRAF Data
Vignettes
Overview Matrix
Tech Groups
Domains
Archetypes
CWRAF FAQs
Other Items of Interest
CWSS
Terms of Use

**CWRAF version:** 0.8.2

**Date:** July 3, 2012

**Project Coordinator:**
Bob Martin (MITRE)

**Document Editor:**
Steve Christey (MITRE)

## CWRAF - Vignette Overview Matrix

This matrix provides an overview of the vignettes that are being actively defined within CWRAF, as summarized based on their domains and technology groups.

| Tech Groups / Business Domains | banking- finance | chemical | ecomm | emerg- svc | energy | evoting | human- res | natl- defense | pub- health | soc- media | telecom |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Web Applications | fin-trade, e-banking | | retail-www | | smart-meter, smart-grid-RUS, smart-grid-gw, reg-elec, scada-hist, web-scada-hmi | elec-abs-int, evoting-Internet, corp-vote | emp-comp | | med-billing, med-device | soc-net, elec-date | tel-ras, web-mail |
| Real-Time Embedded Systems | | | | | smart-meter, smart-grid-RUS, smart-grid-gw | evoting-DRE | | weap-sensor | med-device | | |
| Control Systems | | chem-flow | | | smart-meter, smart-grid-RUS, smart-grid-gw, reg-elec, scada-hist, web-scada-hmi | | | | | | |
| End-Point Computing Devices | | | | first-resp | | | | | med-device | | |
| Database & Storage Systems | e-banking | | retail-www | | scada-hist, web-scada-hmi | evoting-DRE | emp-comp | | med-billing | | |
| Operating Systems | | | retail-www | | web-scada-hmi | elec-abs-int, evoting-Internet, corp-vote | | | med-billing, med-device | | |
| Identity Management Systems | | | | | | | | | | | |
| Enterprise Systems & Applications | e-banking | | retail-www | | scada-hist, web-scada-hmi | elec-abs-int, evoting-Internet, corp-vote | emp-comp | | med-billing, med-device | | |
| Cloud Computing | | | | | | | | | | | |
| Enterprise Security Products | | | | | | | | | | | |
| Network Communications | | | | | web-scada-hmi | evoting-DRE, evoting-Internet | | | | | |

BACK TO TOP

# Domain Summary

This is an up-to-date list of domains as used by CWRAF. For each domain, a list of associated vignettes is provided.

| Domain | Description |
|---|---|
| e-Commerce | The use of the Internet or other computer networks for the sale of products and services, typically using the WWW.<br><br>Vignettes: Web-Based Retail Provider |
| Banking & Finance | Financial industry, including depository financial institutions (banks, thrifts, and credit unions), insurers, securities brokers/dealers, investment companies, some financial utilities, and their associated regulatory systems and agencies.<br><br>Vignettes: Financial Trading, Online Banking |
| Energy | Smart Grid (electrical network through a large region, using digital technology for monitoring or control), nuclear power stations, oil and gas transmission, etc.<br><br>Vignettes: Household Smart Meter, Smart Grid remote utility server, Smart Grid Neighborhood Gateway, Regional Electricity Flow Control, SCADA Historian, Distributed Production Facility Management using SCADA Web-based HMI |
| Chemical | Chemical processing and distribution, etc.<br><br>Vignettes: Chemical Flow Control |
| Manufacturing | Plants and distribution channels, supply chain, etc.<br><br>*No vignettes defined.* |
| Shipping & Transportation | Aerospace (such as safety-critical ground aviation systems, on-board avionics, etc.), highway, maritime transportation, mass transit, pipeline systems, and rail.<br><br>*No vignettes defined.* |
| National Defense | Weapon systems, Intel networks, Defense Industrial Base, etc.<br><br>Vignettes: Weapon system sensor |
| Homeland Security | CBP, Coast Guard, Secret Service, TSA, etc.<br><br>*No vignettes defined.* |

| | |
|---|---|
| Government (Other) | Government (other than National Defense and Homeland Security)<br><br>*No vignettes defined.* |
| Emergency Services | Systems and services that support for First Responders, incident management and response, law enforcement, and emergency services for citizens, etc. The organizations and processes for protecting and preserving critical assets before, during, and after a disaster or catastrophe.<br><br>Vignettes: First Responder |
| Public Health | 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.<br><br>Vignettes: Medical Billing, Human Medical Devices |
| Food & Water | Food processing, clean water treatment and distribution (including dams and processing facilities), etc.<br><br>*No vignettes defined.* |
| Telecommunications | Cellular services, land lines, VOIP, cable & fiber networks, etc.<br><br>Vignettes: Teleworking - Remote Access Server, Teleworking - Web Mail |
| Teleworking | Support for employees to have remote access to internal business networks and capabilities, e.g. networking-capable PDAs and cell phones, VPNs, Network Access Control (NAC), Web-based email services, etc.<br><br>*No vignettes defined.* |
| e-Voting | Electronic voting systems, whether for state-run elections, shareholder meetings, etc.<br><br>Vignettes: State Election Administration using remote Internet voting via absentee ballot, State or Local Elections using eVoting via Direct Recording Election Machines., State or Local Elections using eVoting via an Internet web application, Corporate Shareholder Internet voting |
| Social Media | (Example Domain) The use of the Internet or other computer networks for communication, collaboration, or entertainment in which a large group of users can interact with each other. This includes social networking, wikis, blogs, music and photograph sharing, product/service reviews, bookmarking, etc.<br><br>Vignettes: Social Networking, Electronic Dating |
| Human Resources | (Example Domain) Human resources - management of personnel within an organization, including recruitment, compensation (salary and benefits), performance assessment, training, etc.<br><br>Vignettes: Employee Compensation |

## Vignette Summary

| | |
|---|---|
| banking-finance | |
| Financial Trading | Internet-facing, E-commerce provider of retail goods or services. Data-centric - Database containing PII, credit card numbers, and inventory. |
| Online Banking | The web-based interaction between a bank, credit union, or other financial institution and its consumers for managing accounts, paying bills, and conducting financial transactions. |
| chemical | |
| Chemical Flow Control | A SCADA-based flow control system for a chemical plant. Underlying technology - heavy C usage. Systems developed in pre-Internet era with management consoles interfacing to them. |
| ecomm | |
| Web-Based Retail Provider | Internet-facing, E-commerce provider of retail goods or services. Data-centric - Database containing PII, credit card numbers, and inventory. |
| emerg-svc | |
| First Responder | First responder (such as fire, police, and emergency medical personnel) for a disaster or catastrophe. |
| energy | |
| Household Smart Meter | Meter within the Smart Grid that records electrical consumption and communicates this information to the supplier on a regular basis. |
| Smart Grid remote utility server | Obtains information from smart meters through neighborhood gateways. |
| Smart Grid Neighborhood Gateway | Appliance between smart meter and remote utility server. |
| Regional Electricity Flow Control | Flow control for an electricity network throughout a relatively large region, to further connect suppliers and consumers. Power now enters the grid from both sides (classic provider, but also home-to-provider e.g. home photo-voltaic and wind turbines in homes and throughout the landscape). System needs to have "smarts" to the load leveling capabilities of the grid which is basically a large distributed SCADA-type system. |
| SCADA Historian | Historian server for archival and analysis of data for a SCADA system. Contains a database backend and is accessible via a web interface. Access to the server is typically restricted to a DMZ or internal network. |
| Distributed Production Facility Management using SCADA Web-based HMI | A web-based Human Machine Interface (HMI) for SCADA systems. Users can visualize and control industrial automation processes in real-time from a control interface directly in communication with remote sensors and data collection points. All facets of production can be monitored and managed from a web browser.<br><br>The HMI uses various frameworks (Java, .NET, etc.) with Restful Architecture (AJAX, XML, SOAP, XSL, and WML). |

| | |
|---|---|
| evoting | |
| State Election Administration using remote Internet voting via absentee ballot | Internet-facing polling system supporting high-volume transactions, high availability, Data-centric Database containing ballot information, Audit log generation for each voter. |
| State or Local Elections using eVoting via Direct Recording Election Machines. | DRE systems are not directly connected with the Internet. Vote data is uploaded to a centralized server via modem. Election worker retrieves hardcopies of the voting record from the machine and delivers the printouts to election officials. DRE machines are programmed with firmware uploaded from a compact flash card. It is generally accepted that the computer used to upload the firmware to the flash card should not be connected to the Internet. |
| State or Local Elections using eVoting via an Internet web application | Internet-facing polling systems are connected to the Internet and are designed to support high-volume transactions and high availability. A Data-centric Database is used to collect ballot information, Audit logs are generated for each voter. |
| Corporate Shareholder Internet voting | Corporate Shareholder voting using remote Internet voting |
| human-res | |
| Employee Compensation | Product for managing employee salary and bonuses. PII includes salary, financial transaction (e.g. for direct deposit), social security number, home address, etc. |
| natl-defense | |
| Weapon system sensor | Sensor for a weapons system that is connected to the Global Information Grid (GIG). |
| pub-health | |
| Medical Billing | Medical encoding and billing. Data used includes Electronic Health Records (EHR), financial management, interactions with insurance companies. |
| Human Medical Devices | Medical devices - "implantable" or "partially embedded" in humans, as well as usage in clinic or hospital environments ("patient care" devices.) Includes items such as pacemakers and automatic drug delivery. Control or monitoring of the device might be performed by smartphones. The devices are not in a physically secured environment. |
| soc-media | |
| Social Networking | Web site for enabling a large community of people to post comments, create profiles, exchange messages or pictures, and join affiliation groups, e.g. Facebook, MySpace, Twitter, or LinkedIn. Free-form content, high connectivity between users, private messaging. Heavy Web 2.0 usage. |
| Electronic Dating | Web site for electronic dating. Users can create profiles with pictures, exchange private email, participate in discussion forums, perform searches. Heavy Web 2.0. |
| telecom | |
| Teleworking - Remote Access Server | Remote Access Server used to support employees working outside the enterprise, including teleworking/telecommuting. |
| Teleworking - Web Mail | Use of web-based email for remote access. |

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
FAQs
Community
SwA On-Ramp
T-Shirt
Discussion List
Discussion Archives
Scoring
CWSS
CWRAF
CWE/SANS Top 25
Compatibility
Requirements
Coverage Claims Representation
Compatible Products
Make a Declaration
News
Calendar
Free Newsletter
Contact Us
Search the Site

# Creating Your Own Vignettes

Currently, there are approximately 20 Vignettes and Technical Scorecards, but anyone can create their own Vignette and its accompanying Technical Scorecard to identify which CWEs are most significant to their business and applications. This section will help guide you through that process.

One of the items found in these sample Vignettes is the "Archetypes". A list of the currently defined Archetypes that are available for use in describing Vignettes is here. If there are new Archetypes you need just identify them and send them to cwe@mitre.org and we can add them to the list.

These Archetypes are used as the context for describing the technical elements utilized by the application described in the Vignette.

There are two tables for each Vignette, "Vignette Definition" and "Technical Impact Scorecard".

## Vignette Definition

Creating a Vignette Definition basically comes down to filling in the Vignette Definition table. Below is an example Vignette Definition table with a specific Vignette for a Web-Based Retail Provider described. The Vignette Definition is meant to talk about what business issues are of concern for the application. Is the application dealing with PII? Credit card (PCI-relevant) data? How bad is each of the 8 Technical Impacts given what the application is doing for a business (in the business's operational context).

| | |
|---|---|
| Name | Web-Based Retail Provider |
| ID | retail-www |
| Maturity | under-development |
| Domain | ecomm |
| Desc | Internet-facing, E-commerce provider of retail goods or services. Data-centric - Database |

Section Contents
CWRAF
Introduction
CWSS Scoring in CWRAF
Creating Your Own Vignettes
Future Versions and Activities
Change Log
Vignettes
Tech Groups and Domains
Archetypes
CWRAF FAQs
Other Items of Interest
CWSS
Terms of Use

## CWRAF - Archetypes

Following is a list of the archetypes that are used in CWRAF.

| | |
|---|---|
| Anti-Virus Program | |
| Authentication Server | Teleworking - Remote Access Server, Teleworking - Web Mail |
| B2B Communications | Medical Billing |
| Custom applications | |
| Database | Web-Based Retail Provider, Online Banking, Medical Billing, SCADA Historian, Distributed Production Facility Management using SCADA Web-based HMI, Employee Compensation |
| Development Framework | State or Local Elections using eVoting via an Internet web application |
| Digital certificate | |
| Distributed Control System (DCS) | |
| Document Processing | |
| Embedded Device | Human Medical Devices, Household Smart Meter, Smart Grid remote utility server, Smart Grid Neighborhood Gateway, State or Local Elections using eVoting via Direct Recording Election Machines., Weapon system sensor |
| Endpoint System | Distributed Production Facility Management using SCADA Web-based HMI, State or Local Elections using eVoting via Direct Recording Election Machines. |
| Firewall | |
| General-purpose OS | Web-Based Retail Provider, Medical Billing, Human Medical Devices, Distributed Production Facility Management using SCADA Web-based HMI, State Election Administration using remote Internet voting via absentee ballot, State or Local Elections using eVoting via an Internet web application, Corporate Shareholder Internet voting |
| Infrastructure as a Service (IaaS) | |
| Internet Communications | Distributed Production Facility Management using SCADA Web-based HMI, State or Local Elections using eVoting via an Internet web application |
| J2EE and supporting frameworks | Financial Trading |
| Laptop | |
| Modem Communications | State or Local Elections using eVoting via Direct Recording Election Machines. |
| N-tier distributed | Financial Trading |
| PDA | |

| | |
|---|---|
| PKI | |
| Platform-as-a-Service (PaaS) | |
| Privacy management | |
| Process Control Systems | Household Smart Meter, Smart Grid remote utility server, Smart Grid Neighborhood Gateway, Regional Electricity Flow Control, SCADA Historian, Distributed Production Facility Management using SCADA Web-based HMI, Chemical Flow Control |
| Programmable Logic Controller (PLC) | |
| Proprietary Firmware | State or Local Elections using eVoting via Direct Recording Election Machines. |
| Remote Terminal Unit (RTU) | |
| Removable Storage Media | State or Local Elections using eVoting via Direct Recording Election Machines. |
| Router | |
| SCADA | |
| SOA-based web service | |
| Service-oriented architecture | Social Networking, Electronic Dating |
| Smartphone | Human Medical Devices, First Responder |
| Software-as-a-Service (SaaS) | |
| Transactional engine | Financial Trading, Online Banking |
| VPN | |
| Virtualized OS | |
| Web application | Distributed Production Facility Management using SCADA Web-based HMI, State or Local Elections using eVoting via an Internet web application |
| Web browser | Web-Based Retail Provider, Online Banking, Medical Billing, Distributed Production Facility Management using SCADA Web-based HMI, State Election Administration using remote Internet voting via absentee ballot, State or Local Elections using eVoting via an Internet web application, Corporate Shareholder Internet voting, Social Networking, Electronic Dating, Employee Compensation, Teleworking - Remote Access Server, Teleworking - Web Mail |
| Web browser plugin | |
| Web client | Human Medical Devices, Household Smart Meter, Smart Grid remote utility server, Smart Grid Neighborhood Gateway, Regional Electricity Flow Control, SCADA Historian |
| Web proxy | |
| Web server | Web-Based Retail Provider, Online Banking, Medical Billing, Regional Electricity Flow Control, SCADA Historian, Distributed Production Facility Management using SCADA Web-based HMI, State Election Administration using remote Internet voting via absentee ballot, Corporate Shareholder Internet voting, Social Networking, Electronic Dating, Employee Compensation, Teleworking - Remote Access Server, Teleworking - Web Mail |
| Web service | Distributed Production Facility Management using SCADA Web-based HMI |

NIST

Software Assurance Metrics and Tool Evaluation (SAMATE)

10001
011111
10001
111110
10001
SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity


Vulnerability Analysis of Energy Delivery Control Systems

INL
Idaho National Laboratory

CWE™

IARPA

Securely Taking On New Executable Software of Uncertain Provenance (STONESOUP)

# ISO/IEC JTC 1/SC 27/WG 3, NWP

## Refining Software Vulnerability Analysis Under ISO/IEC 15049 and ISO/IEC 18045

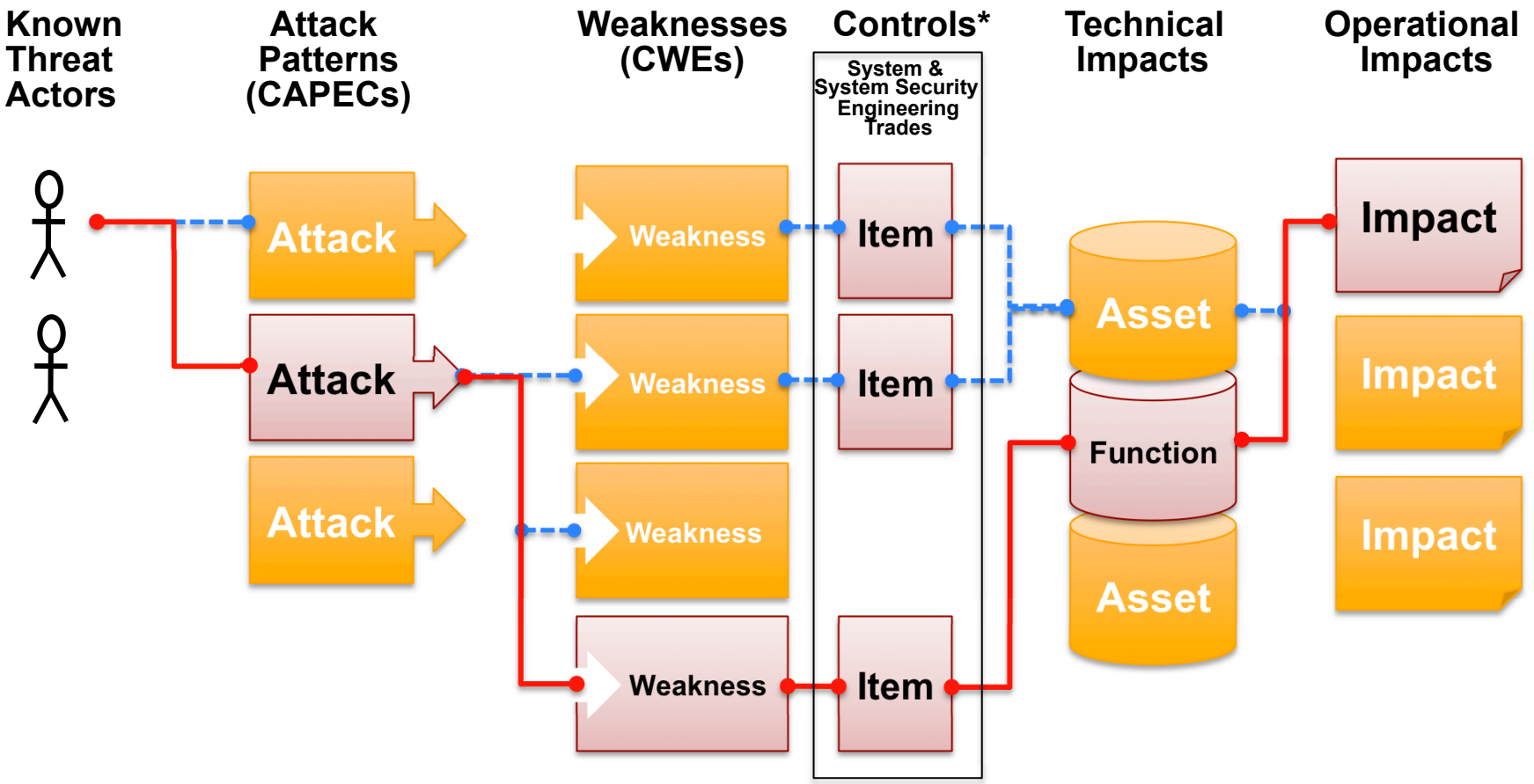
- The way how the CAPEC and related CWE taxonomies are to be used by the developer, which needs to consider and provide sufficient and effective mitigation to all applicable attacks and weaknesses.

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

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

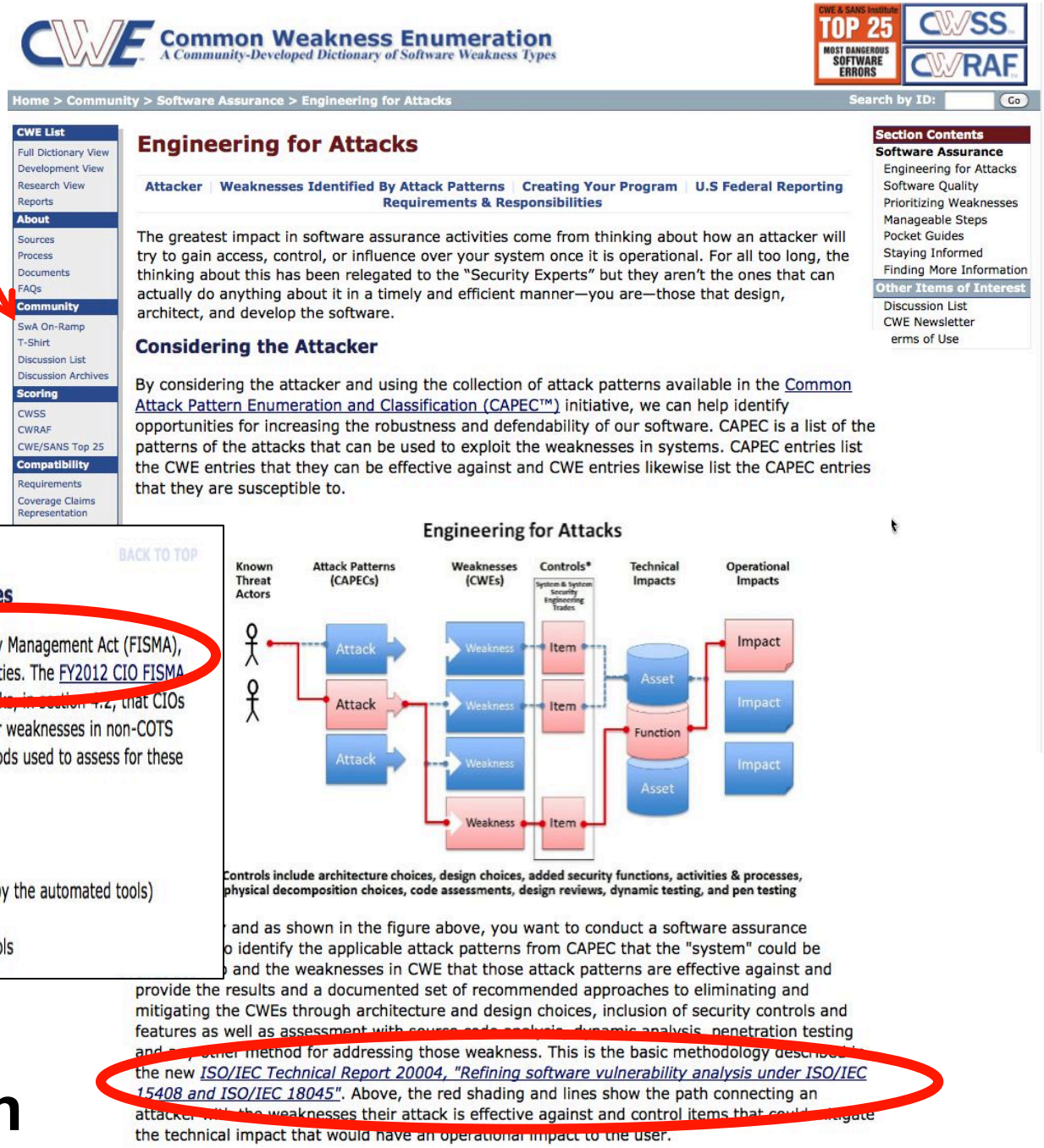
- How to incorporate to the evaluation attacks and weaknesses not included in the CAPEC.

# What Are the System Security Risks?




* Controls include architecture choices, design choices, added security functions, activities & processes, physical decomposition choices, code assessments, design reviews, dynamic testing, and pen testing

# New "SwA On-Ramp" In the CWE Community section of the web site



CWE Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

CWE & SANS Institute TOP 25 MOST DANGEROUS SOFTWARE ERRORS | CWSS | CWRAF

Home > Community > Software Assurance > Engineering for Attacks

Search by ID: Go

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
FAQs
Community
SwA On-Ramp
T-Shirt
Discussion List
Discussion Archives
Scoring
CWSS
CWRAF
CWE/SANS Top 25
Compatibility
Requirements
Coverage Claims Representation

## Engineering for Attacks

Attacker | Weaknesses Identified By Attack Patterns | Creating Your Program | U.S Federal Reporting Requirements & Responsibilities

The greatest impact in software assurance activities come from thinking about how an attacker will try to gain access, control, or influence over your system once it is operational. For all too long, the thinking about this has been relegated to the "Security Experts" but they aren't the ones that can actually do anything about it in a timely and efficient manner—you are—those that design, architect, and develop the software.

Section Contents
Software Assurance
Engineering for Attacks
Software Quality
Prioritizing Weaknesses
Manageable Steps
Pocket Guides
Staying Informed
Finding More Information
Other Items of Interest
Discussion List
CWE Newsletter
erms of Use

### Considering the Attacker

By considering the attacker and using the collection of attack patterns available in the Common Attack Pattern Enumeration and Classification (CAPEC™) initiative, we can help identify opportunities for increasing the robustness and defendability of our software. CAPEC is a list of the patterns of the attacks that can be used to exploit the weaknesses in systems. CAPEC entries list the CWE entries that they can be effective against and CWE entries likewise list the CAPEC entries that they are susceptible to.

BACK TO TOP

### U.S Federal Reporting Requirements & Responsibilities

The U.S. Federal Government, under the Federal Information Security Management Act (FISMA), must now report some specifics about their software assurance activities. The FY2012 CIO FISMA Reporting Metrics issued by DHS's Federal Network Security (FNS) asks, in section 4.2, that CIOs utilize CWE, CAPEC and CWSS to discuss what was done to search for weaknesses in non-COTS before release. The document also asks for a description of the methods used to assess for these issues, offering several possibilities. Specifically:

- Web scanners for web- based applications
- Static Code Analysis Tools
- Manual code reviews (especially for weaknesses not covered by the automated tools)
- Dynamic Code Analysis Tools
- PEN testing for attack types not covered by the automated tools

Engineering for Attacks




Controls include architecture choices, design choices, added security functions, activities & processes, physical decomposition choices, code assessments, design reviews, dynamic testing, and pen testing

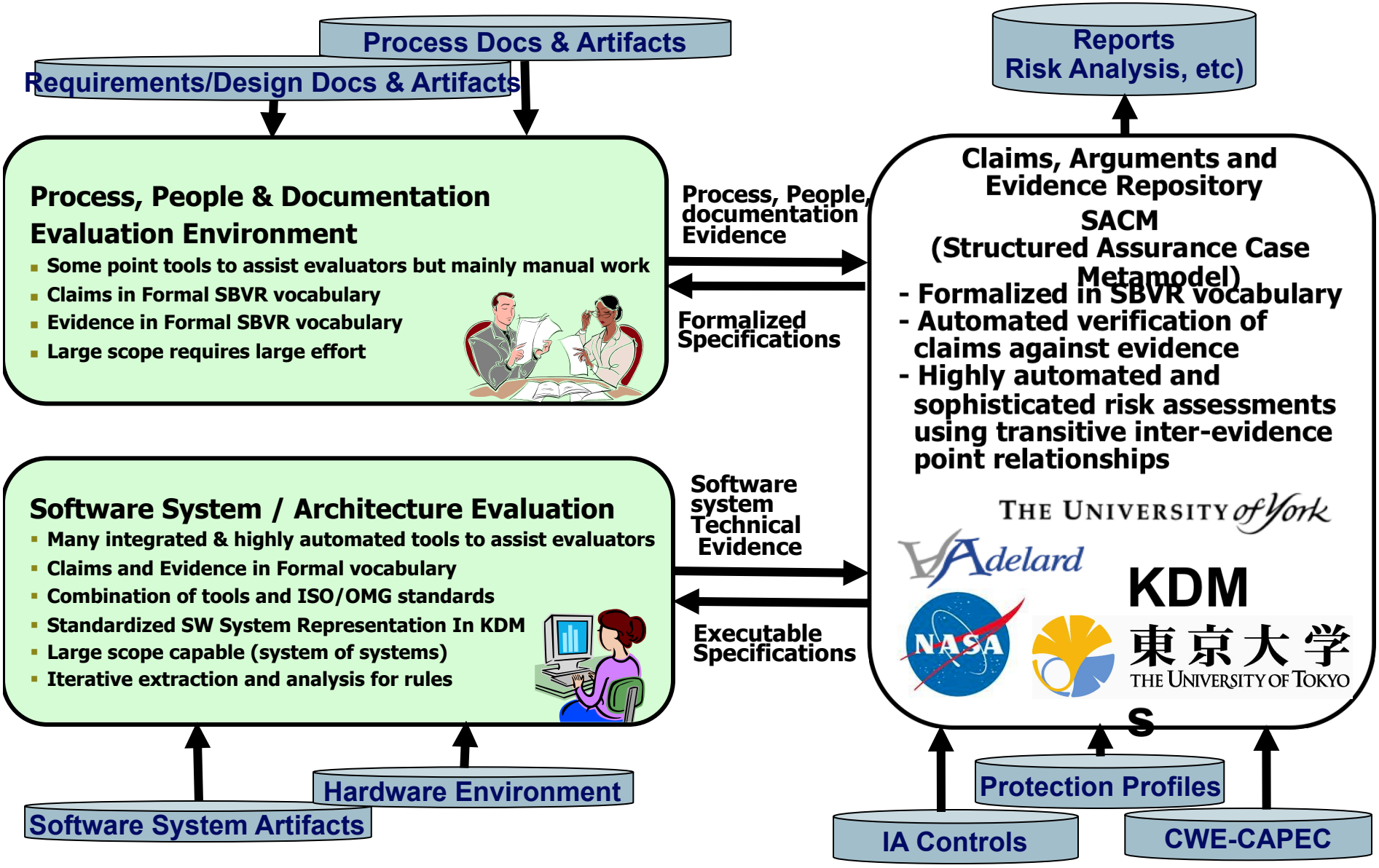
and as shown in the figure above, you want to conduct a software assurance
o identify the applicable attack patterns from CAPEC that the "system" could be
and the weaknesses in CWE that those attack patterns are effective against and
provide the results and a documented set of recommended approaches to eliminating and
mitigating the CWEs through architecture and design choices, inclusion of security controls and
features as well as assessment with source code analysis, dynamic analysis, penetration testing
and any other method for addressing those weakness. This is the basic methodology described in
the new *ISO/IEC Technical Report 20004, "Refining software vulnerability analysis under ISO/IEC 15408 and ISO/IEC 18045"*. Above, the red shading and lines show the path connecting an
attacker with the weaknesses their attack is effective against and control items that could mitigate
the technical impact that would have an operational impact to the user.

# Introduces DoD's Program Protection Plan (PPP)

**https://cwe.mitre.org/community/swa/attacks.html**

# Software Assurance Ecosystem: The Formal Framework

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



# Summary…

- *An adversary's methods of attack and the system's susceptibility to the attacks that endanger the mission are those to focus mitigations/security capabilities against.*
- *Clear articulation of the threat actors attacks, the weaknesses they can exploit, the mission impacts, and their mitigations through all of the SSE and SE methods, needs to be iteratively worked as a community from the point of system concept through sustainment*

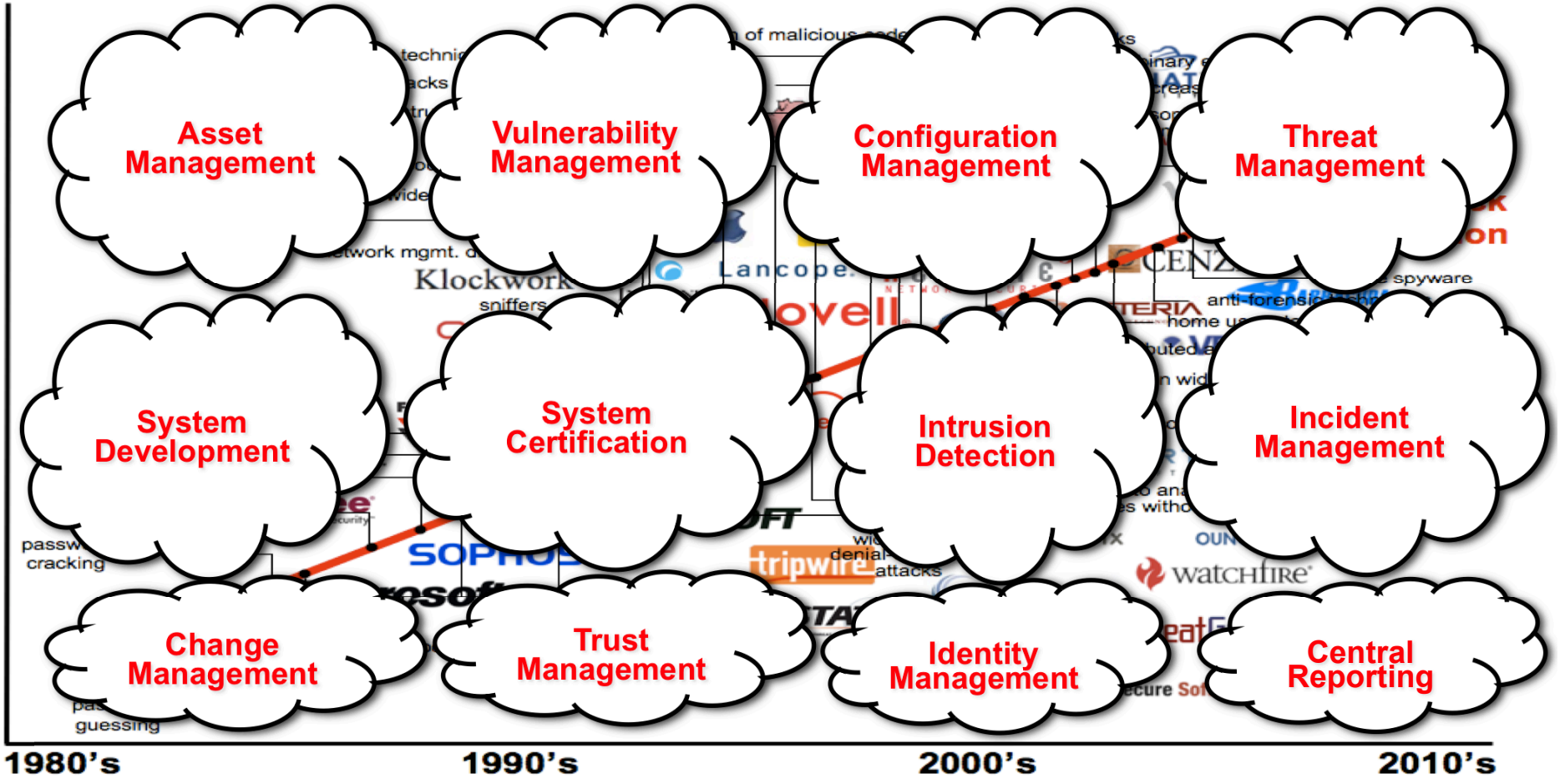
**CAPEC – CWE – CWRAF/CWSS**

**Assurance Case – Structured Assurance Case Metamodel**

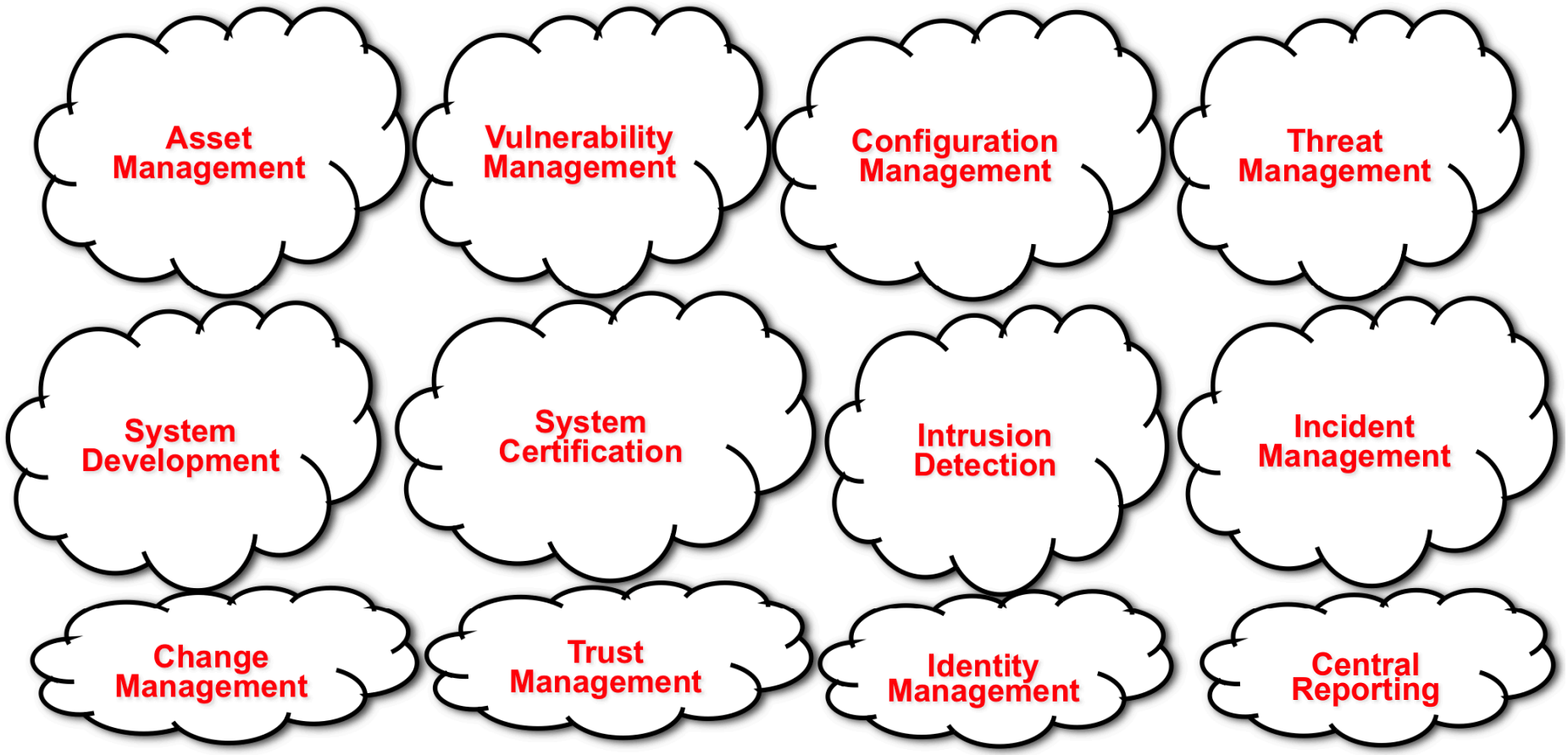
# What Should an App "have" to be in a DoD App Store

- Statement of the SDK used
- Statement of the permissions/capabilities/intents needed by the app
- Statements about the PPP-related facts:
  - **What CVEs – discuss the commercial and open source usage in the app and what is being done to manage vulnerabilities in those items**
  - **What CWEs – discuss the weaknesses that were the focus of static and manual assessments and the tools used**
  - **What CAPECs – discuss the attack patterns used to review the design and architecture of the app and describe the attack patterns used to derive test cases for assessing the app**

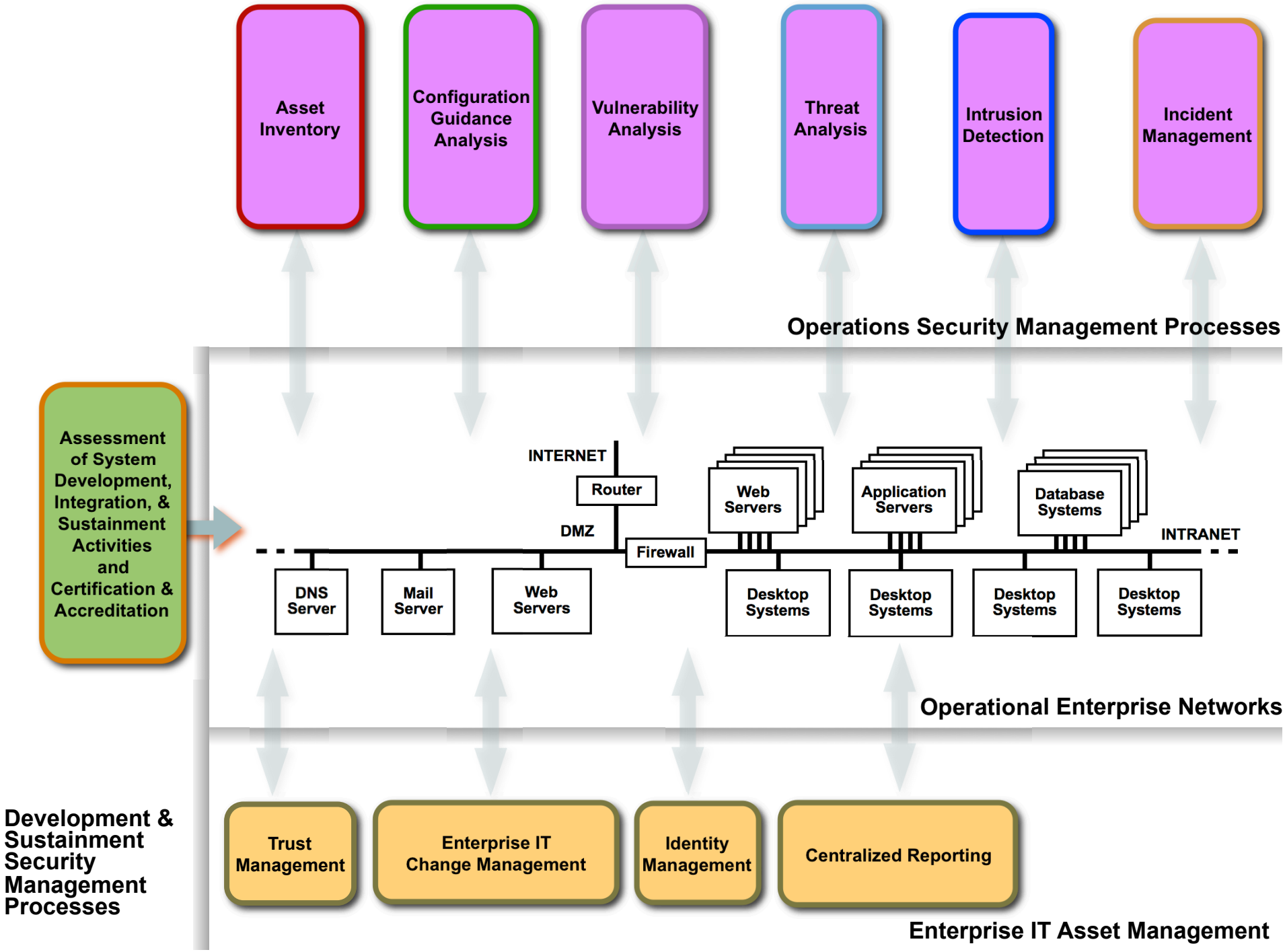
# Architecting Security with Information Standards for COIs

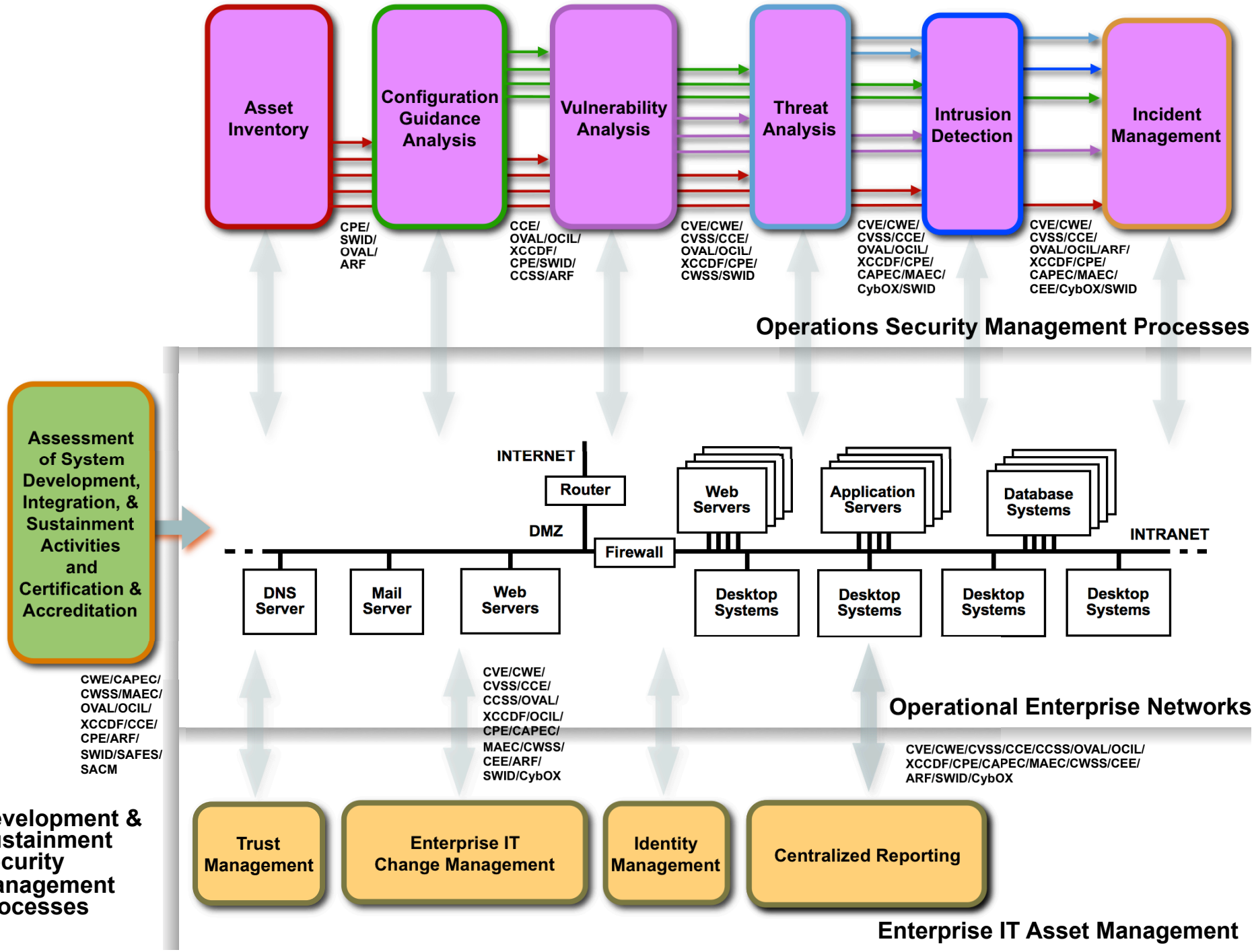



Making Security Measurable™




Making Security Measurable™


Asset Inventory
Configuration Guidance Analysis
Vulnerability Analysis
Threat Analysis
Intrusion Detection
Incident Management
Operations Security Management Processes
Assessment of System Development, Integration, & Sustainment Activities and Certification & Accreditation
INTERNET
Router
DMZ
Firewall
Web Servers
Application Servers
Database Systems
INTRANET
DNS Server
Mail Server
Web Servers
Desktop Systems
Desktop Systems
Desktop Systems
Desktop Systems
Operational Enterprise Networks
Development & Sustainment Security Management Processes
Trust Management
Enterprise IT Change Management
Identity Management
Centralized Reporting
Enterprise IT Asset Management


Asset Inventory
Configuration Guidance Analysis
Vulnerability Analysis
Threat Analysis
Intrusion Detection
Incident Management
CPE/ SWID/ OVAL/ ARF
CCE/ OVAL/OCIL/ XCCDF/ CPE/SWID/ CCSS/ARF
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ XCCDF/CPE/ CWSS/SWID
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ XCCDF/CPE/ CAPEC/MAEC/ CybOX/SWID
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ARF/ XCCDF/CPE/ CAPEC/MAEC/ CEE/CybOX/SWID
Operations Security Management Processes
Assessment of System Development, Integration, & Sustainment Activities and Certification & Accreditation
CWE/CAPEC/ CWSS/MAEC/ OVAL/OCIL/ XCCDF/CCE/ CPE/ARF/ SWID/SAFES/ SACM
INTERNET
Router
DMZ
Firewall
Web Servers
Application Servers
Database Systems
INTRANET
DNS Server
Mail Server
Web Servers
Desktop Systems
Desktop Systems
Desktop Systems
Desktop Systems
CVE/CWE/ CVSS/CCE/ CCSS/OVAL/ XCCDF/OCIL/ CPE/CAPEC/ MAEC/CWSS/ CEE/ARF/ SWID/CybOX
Operational Enterprise Networks
CVE/CWE/CVSS/CCE/CCSS/OVAL/OCIL/ XCCDF/CPE/CAPEC/MAEC/CWSS/CEE/ ARF/SWID/CybOX
Development & Sustainment Security Management Processes
Trust Management
Enterprise IT Change Management
Identity Management
Centralized Reporting
Enterprise IT Asset Management

# Cyber Ecosystem Standardization Efforts

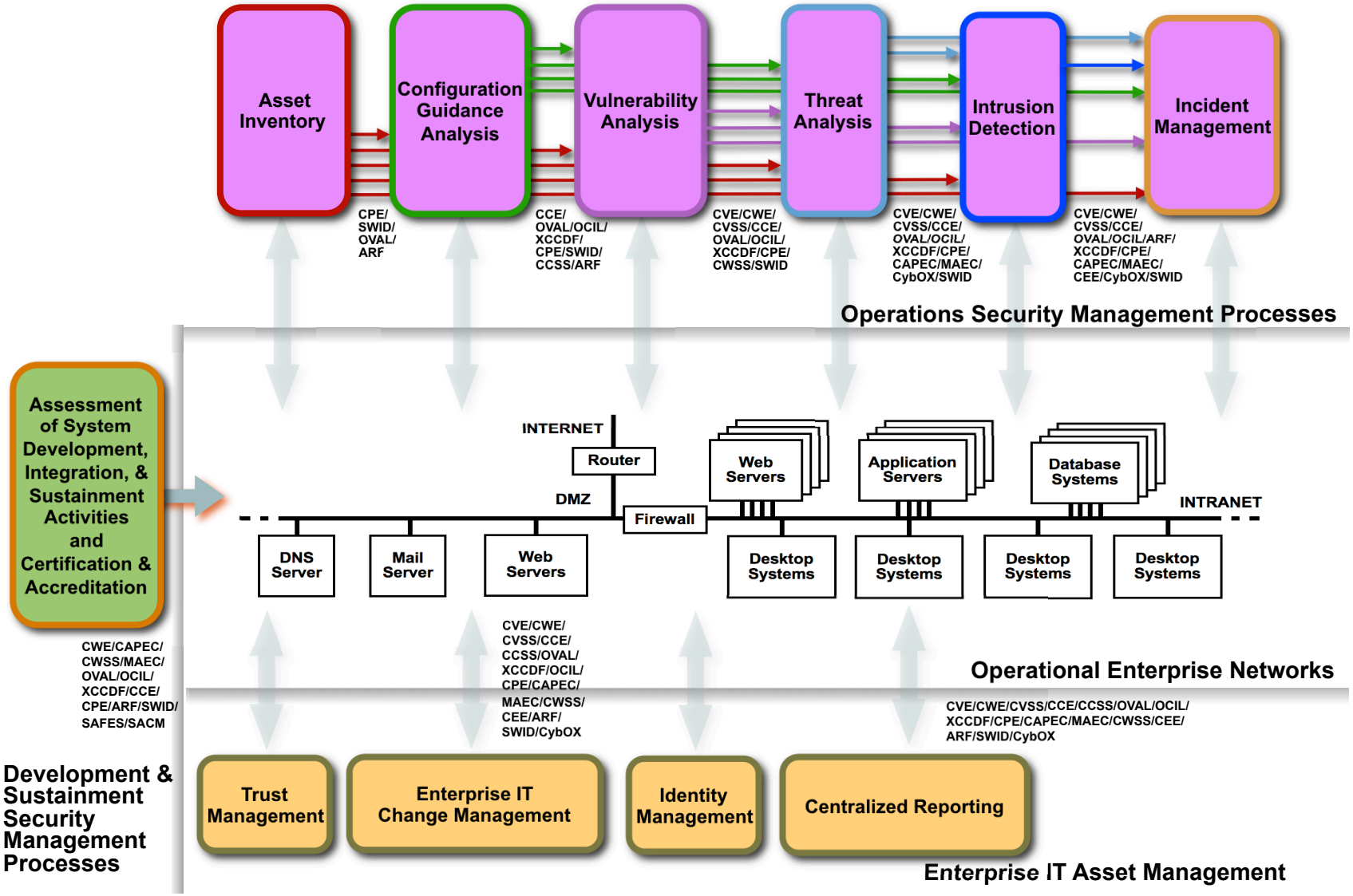
| | |
|---|---|
| What IT systems do I have in my enterprise? | • CPE (Platforms) |
| What known vulnerabilities do I need to worry about? | • CVE (Vulnerabilities) |
| What vulnerabilities do I need to worry about right now? | • CVSS (Scoring System) |
| How can I configure my systems more securely? | • CCE (Configurations) |
| How do I define a policy of secure configurations? | • XCCDF (Configuration Checklists) |
| How can I be sure my systems conform to policy? | • OVAL (Assessment Language) |
| How can I be sure the operation of my systems conforms to policy? | • OCIL (Interactive Language) |
| What weaknesses in my software could be exploited? | • CWE (Weaknesses) |
| What attacks can exploit which weaknesses? | • CAPEC (Attack Patterns) |
| How can we recognize malware & share that info? | • MAEC (Malware Attributes) |
| What observable behavior might put my enterprise at risk? | • CybOX (Cyber Observables) |
| What events should be logged, and how? | • CEE (Events) |
| How can I aggregate assessment results? | • ARF (Assessment Results) |

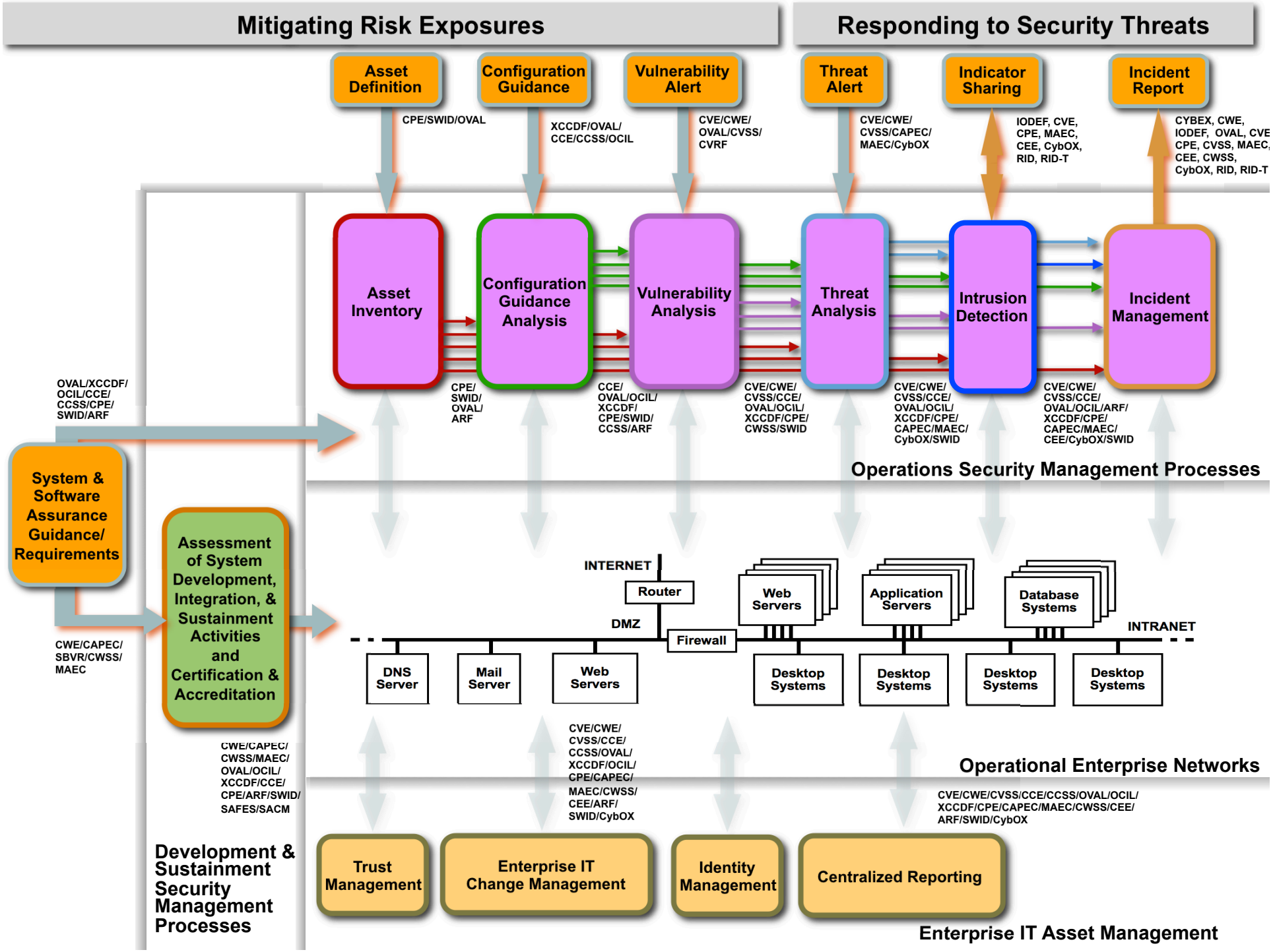
# Standardization Efforts leveraged by the Security Content Automation Protocol (SCAP)

| | |
|---|---|
| What IT systems do I have in my enterprise? | • CPE (Platforms) |
| What known vulnerabilities do I need to worry about? | • CVE (Vulnerabilities) |
| What vulnerabilities do I need to worry about right now? | • CVSS (Scoring System) |
| How can I configure my systems more securely? | • CCE (Configurations) |
| How do I define a policy of secure configurations? | • XCCDF (Configuration Checklists) |
| How can I be sure my systems conform to policy? | • OVAL (Assessment Language) |
| How can I be sure the operation of my systems conforms to policy? | • OCIL (Interactive Language) |
| What weaknesses in my software could be exploited? | • CWE (Weaknesses) |
| What attacks can exploit which weaknesses? | • CAPEC (Attack Patterns) |
| How can we recognize malware & share that info? | • MAEC (Malware Attributes) |
| What observable behavior might put my enterprise at risk? | • CybOX (Cyber Observables) |
| What events should be logged, and how? | • CEE (Events) |
| How can I aggregate assessment results? | • ARF (Assessment Results) |

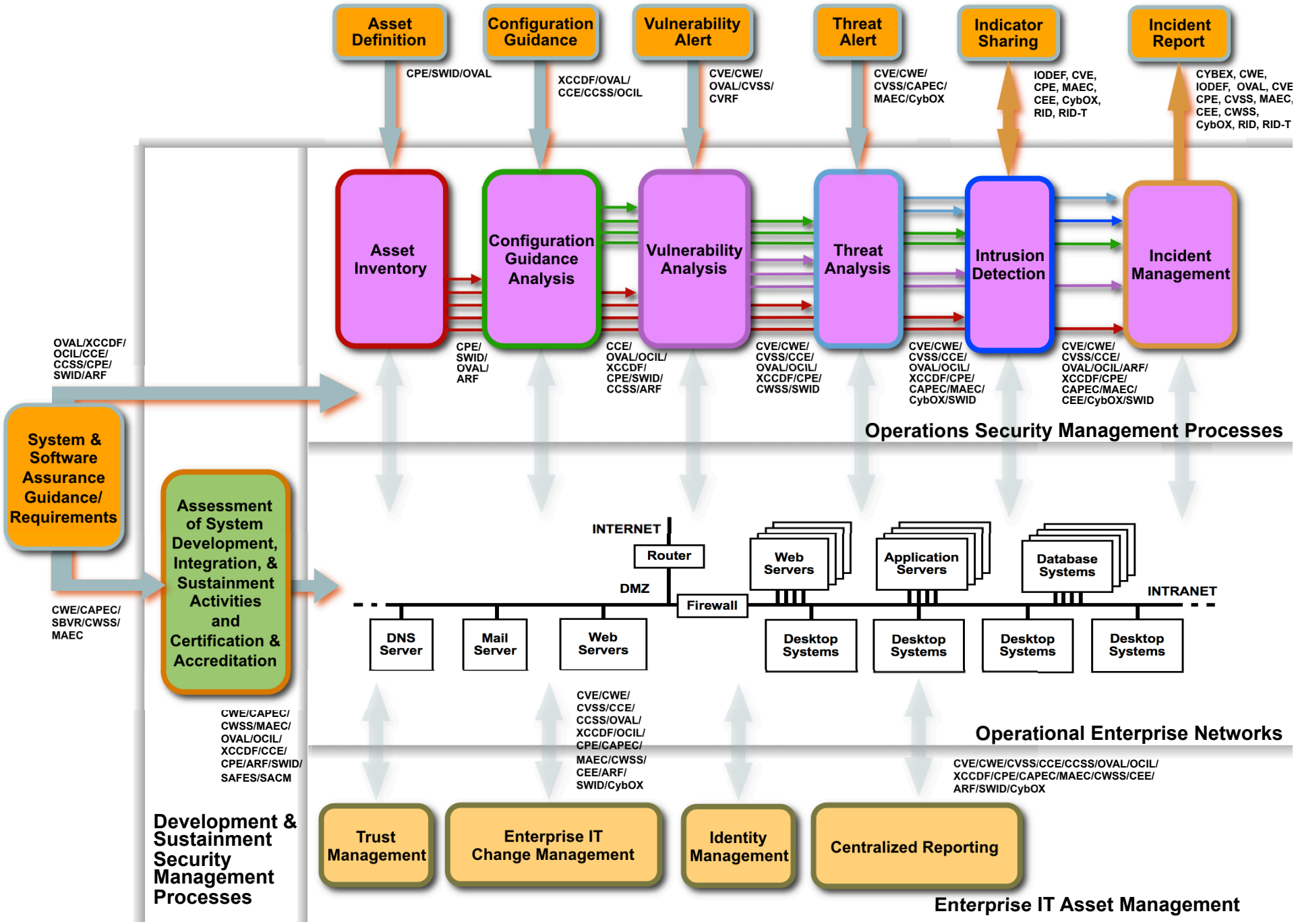
# Standardization Efforts focused on mitigating risks and enabling faster incident response

| | |
|---|---|
| What IT systems do I have in my enterprise? | • CPE (Platforms) |
| What known vulnerabilities do I need to worry about? | • CVE (Vulnerabilities) |
| What vulnerabilities do I need to worry about right now? | • CVSS (Scoring System) |
| How can I configure my systems more securely? | • CCE (Configurations) |
| How do I define a policy of secure configurations? | • XCCDF (Configuration Checklists) |
| How can I be sure my systems conform to policy? | • OVAL (Assessment Language) |
| How can I be sure the operation of my systems conforms to policy? | • OCIL (Interactive Language) |
| What weaknesses in my software could be exploited? | • CWE (Weaknesses) |
| What attacks can exploit which weaknesses? | • CAPEC (Attack Patterns) |
| How can we recognize malware & share that info? | • MAEC (Malware Attributes) |
| What observable behavior might put my enterprise at risk? | • CybOX (Cyber Observables) |
| What events should be logged, and how? | • CEE (Events) |
| How can I aggregate assessment results? | • ARF (Assessment Results) |


Asset Inventory
Configuration Guidance Analysis
Vulnerability Analysis
Threat Analysis
Intrusion Detection
Incident Management
CPE/ SWID/ OVAL/ ARF
CCE/ OVAL/OCIL/ XCCDF/ CPE/SWID/ CCSS/ARF
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ XCCDF/CPE/ CWSS/SWID
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ XCCDF/CPE/ CAPEC/MAEC/ CybOX/SWID
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ARF/ XCCDF/CPE/ CAPEC/MAEC/ CEE/CybOX/SWID
Operations Security Management Processes
Assessment of System Development, Integration, & Sustainment Activities and Certification & Accreditation
CWE/CAPEC/ CWSS/MAEC/ OVAL/OCIL/ XCCDF/CCE/ CPE/ARF/SWID/ SAFES/SACM
INTERNET
Router
DMZ
Firewall
INTRANET
Web Servers
Application Servers
Database Systems
DNS Server
Mail Server
Web Servers
Desktop Systems
Desktop Systems
Desktop Systems
Desktop Systems
CVE/CWE/ CVSS/CCE/ CCSS/OVAL/ XCCDF/OCIL/ CPE/CAPEC/ MAEC/CWSS/ CEE/ARF/ SWID/CybOX
CVE/CWE/CVSS/CCE/CCSS/OVAL/OCIL/ XCCDF/CPE/CAPEC/MAEC/CWSS/CEE/ ARF/SWID/CybOX
Operational Enterprise Networks
Development & Sustainment Security Management Processes
Trust Management
Enterprise IT Change Management
Identity Management
Centralized Reporting
Enterprise IT Asset Management


Mitigating Risk Exposures
Responding to Security Threats
Asset Definition
Configuration Guidance
Vulnerability Alert
Threat Alert
Indicator Sharing
Incident Report
CPE/SWID/OVAL
XCCDF/OVAL/ CCE/CCSS/OCIL
CVE/CWE/ OVAL/CVSS/ CVRF
CVE/CWE/ CVSS/CAPEC/ MAEC/CybOX
IODEF, CVE, CPE, MAEC, CEE, CybOX, RID, RID-T
CYBEX, CWE, IODEF, OVAL, CVE CPE, CVSS, MAEC, CEE, CWSS, CybOX, RID, RID-T
Asset Inventory
Configuration Guidance Analysis
Vulnerability Analysis
Threat Analysis
Intrusion Detection
Incident Management
OVAL/XCCDF/ OCIL/CCE/ CCSS/CPE/ SWID/ARF
CPE/ SWID/ OVAL/ ARF
CCE/ OVAL/OCIL/ XCCDF/ CPE/SWID/ CCSS/ARF
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ XCCDF/CPE/ CWSS/SWID
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ XCCDF/CPE/ CAPEC/MAEC/ CybOX/SWID
CVE/CWE/ CVSS/CCE/ OVAL/OCIL/ARF/ XCCDF/CPE/ CAPEC/MAEC/ CEE/CybOX/SWID
Operations Security Management Processes
System & Software Assurance Guidance/ Requirements
Assessment of System Development, Integration, & Sustainment Activities and Certification & Accreditation
CWE/CAPEC/ SBVR/CWSS/ MAEC
INTERNET
Router
DMZ
Firewall
INTRANET
Web Servers
Application Servers
Database Systems
DNS Server
Mail Server
Web Servers
Desktop Systems
Desktop Systems
Desktop Systems
Desktop Systems
CWE/CAPEC/ CWSS/MAEC/ OVAL/OCIL/ XCCDF/CCE/ CPE/ARF/SWID/ SAFES/SACM
CVE/CWE/ CVSS/CCE/ CCSS/OVAL/ XCCDF/OCIL/ CPE/CAPEC/ MAEC/CWSS/ CEE/ARF/ SWID/CybOX
Operational Enterprise Networks
CVE/CWE/CVSS/CCE/CCSS/OVAL/OCIL/ XCCDF/CPE/CAPEC/MAEC/CWSS/CEE/ ARF/SWID/CybOX
Development & Sustainment Security Management Processes
Trust Management
Enterprise IT Change Management
Identity Management
Centralized Reporting
Enterprise IT Asset Management

## Knowledge Repositories






# Contact Info

**cwe@mitre.org**
**capec@mitre.org**
**cwss@mitre.org**
**msm@mitre.org**



# SOFTWARE ASSURANCE FORUM

Homeland Security

Commerce

National Defense

## BUILDING SECURITY IN


**Public/Private Collaboration Efforts for Security Automation and Software Supply Chain Risk Management**

**Next SwA Forum meets 18-20 Sep 2012 at MITRE, McLean, VA**

# End – Part 2