Software & Supply Chain Assurance:

Enabling Enterprise Resilience through Security Automation, Software Assurance, and Supply Chain Risk Management

Joe Jarzombek, PMP, CSSLP
Director for Software & Supply Chain Assurance
Stakeholder Engagement & Cyber Infrastructure Resilience
Cyber Security & Communications
Public-Private Collaboration Efforts for Security Automation, Software Assurance, and Supply Chain Risk Management
DHS Cybersecurity and Communications

Responsible for enhancing security, resiliency, and reliability of the nation's cyber and communications infrastructure; actively engages the public and private sectors as well as international partners to prepare for, prevent, and respond to catastrophic incidents that could degrade or overwhelm strategic assets.

Works to prevent or minimize disruptions to our critical information infrastructure in order to protect the public, the economy, government services, and overall security of the United States by supporting a series of continuous efforts designed to further safeguard federal government systems by reducing potential vulnerabilities, protecting against cyber intrusions, and anticipating future threats.

As the Sector-Specific Agency for the Communications and Information Technology (IT) sectors, CS&C coordinates national-level reporting consistent with the National Response Framework, and fulfills the mission through its five divisions:

- Office of Emergency Communications (OEC)
- Network Security Deployment (NSD)
- Federal Network Resilience (FNR)
- National Cybersecurity & Communications Integration Center (NCCIC)
- Stakeholder Engagement & Cyber Infrastructure Resilience (SECIR)
DHS Software & Supply Chain Assurance Strategy

Support Response

Enable Automation

Outreach & Collaborate

Manage Software & Supply Chain Risks

Influence Policy

Influence Standards

Policies implement standards

Standards are a foundation of good policy

Tools allow standards and policies to work

Security Automation, Acquisition Risk Analysis/Modeling

Policy decisions influence tool design

Standards are implemented by tools

NCCIC, US-CERT

DOJ, I&A

NPPD Contract Reviews

CFIUS Review, CRR

Etc.

Publications, Websites, FedVTE, Training & Education

Working Groups, Forums

ISO/IEC JTC1

The Open Group

ITU-T CYBEX

OMG

NIST Spec Pubs

SP800-161 & 53

US Fed

DHS Mgmt

& IT Acquisition

Stakeholder Engagement and Cyber Infrastructure Resilience
Gaining confidence in ICT/software-based cyber technologies

- Dependencies on technology are greater than ever
- Possibility of disruption is greater than ever because hardware/software is vulnerable
- Loss of confidence alone can lead to stakeholder actions that disrupt critical business activities
In an era riddled with asymmetric cyber attacks, claims about system reliability and safety must include provisions for built-in security of the enabling software.

High Reliance on ICT/Software

Built-in Security enables Resilience

Critical security controls aligned with mission
Automated continuous diagnostics and mitigation
Everything’s Connected

Your System is attackable…

When this Other System gets subverted through an un-patched vulnerability, an application weakness, or a mis-configuration…
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)

Exploitable Software Weaknesses (CWEs) are sources for future Zero-Day Attacks
Software Assurance Addresses Exploitable Software: Outcomes of non-secure practices and/or malicious intent

Exploitation potential of vulnerability is independent of “intent”

Defects
Unintentional Vulnerabilities
Intentional Vulnerabilities
Malware

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

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.*

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

From CNSS Instruction 4009 “National Information Assurance Glossary” (26APR2010)
ICT/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

Software & Supply Chain Assurance provides a focus for:
-- Resilient Software and ICT Components,
-- Security in the Component Life Cycle,
-- Software Security in Services, and
-- Supply Chain Risk Management (mitigating risks of counterfeit & tainted products)
Increased risk from supply chain due to:

- Increasing dependence on commercial ICT for mission critical systems
- Increasing reliance on globally-sourced ICT hardware, software, and services
  - Varying levels of development/outsourcing controls
  - Lack of transparency in process chain of custody
  - Varying levels of acquisition ‘due-diligence’
- Residual risk passed to end-user enterprise
  - Defective and Counterfeit products
  - Tainted products with malware, exploitable weaknesses and vulnerabilities
- Growing technological sophistication among our adversaries
  - Internet enables adversaries to probe, penetrate, and attack us remotely
  - Supply chain attacks can exploit products and processes throughout the lifecycle
Risk Management (Enterprise ↔ Project): Shared Processes & Practices ↔ Different Focuses

► Enterprise-Level:
  - Regulatory compliance
  - Changing threat environment
  - Business Case

► Program/Project-Level:
  - Cost
  - Schedule
  - Performance

Who makes risk decisions?
Who “owns” residual risk from tainted/counterfeit products?

* “Tainted” products are those that are corrupted with malware, or exploitable weaknesses & vulnerabilities that put users at risk
We Have a Problem

- When the government purchases products or services with inadequate in-built “cybersecurity,” the risks created persist throughout the lifespan of the item purchased. The lasting effect of inadequate cybersecurity in acquired items is part of what makes acquisition reform so important to achieving cybersecurity and resiliency.

- Currently, government and contractors use varied and nonstandard practices, which make it difficult to consistently manage and measure acquisition cyber risks across different organizations.

- Meanwhile, due to the growing sophistication and complexity of ICT and the global ICT supply chains, federal agency information systems are increasingly at risk of compromise, and agencies need guidance to help manage ICT supply chain risks.
Executive Order 13636

- Section 8(e) of the EO required GSA and DoD to:
  “… make recommendations to the President, … on the feasibility, security benefits, and relative merits of incorporating security standards into acquisition planning and contract administration”

- Report signed January 23, 2014 ([http://gsa.gov/portal/content/176547](http://gsa.gov/portal/content/176547))

- Recommends six acquisition reforms:
  I. Institute Baseline Cybersecurity Requirements as a Condition of Contract Award for Appropriate Acquisitions
  II. Address Cybersecurity in Relevant Training
  III. Develop Common Cybersecurity Definitions for Federal Acquisitions
  IV. Institute a Federal Acquisition Cyber Risk Management Strategy
  V. Include a Requirement to Purchase from Original Equipment Manufacturers, Their Authorized Resellers, or Other “Trusted” Sources, Whenever Available, in Appropriate Acquisitions
  VI. Increase Government Accountability for Cyber Risk Management
Gaps

Updated: NIST SP 800-53 – now revision 4

- Expanded focus on System and Services Acquisition (SA) family and other areas such as CM and System Integration
- Overlays (business/mission, environment, etc.)
- Tailor to criticality, threat, responsiveness, system maturity
- Eliminate class of control (Management, Operational, Technical)
A New Approach for Information Security

- Work directly with executives, mission/business owners and program managers.
- Bring all stakeholders to the table with a vested interest in the success or outcome of the mission or business function.
- Consider information security requirements as mainstream functional requirements.
- Conduct security trade-off analyses with regard to cost, schedule, and performance requirements.
- Implement enforceable metrics for key executives.
Increasing Strength - 1

Increasing Strength of IT Infrastructure

- Simplify.
  - Reduce and manage *complexity* of IT infrastructure.
  - Use enterprise architecture to streamline the IT infrastructure; *standardize, optimize, consolidate* IT assets.

- Specialize.
  - Use guidance in SP 800-53, Rev 4 to *customize security plans* to support specific missions/business functions, environments of operation, and technologies.
  - Develop effective *monitoring strategies* linked to specialized security plans.
Increasing Strength of IT Infrastructure

- Integrate.
  - Build information security requirements and controls into mainstream organizational processes including:
    - Enterprise Architecture.
    - Systems Engineering.
    - System Development Life Cycle.
    - Acquisition.
  - Eliminate information security programs and practices as stovepipes within organizations.
  - Ensure information security decisions are risk-based and part of routine cost, schedule, and performance tradeoffs.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Gaps Addressed in Revision 3

Gap Areas Addressed

- Insider threat
- Application security
- Supply chain risk
- Security assurance and trustworthy systems
- Mobile and cloud computing technologies
- Advanced persistent threat
- Tailoring guidance and overlays
- Privacy
It’s not the *number* of security controls that matters…

It’s having the *right* controls to do the job.
Tiers

**STRATEGIC (EXECUTIVE) RISK FOCUS**

Communicating and sharing risk-related information from the **strategic to tactical level**, that is from the **executives to the operators**.

**TIER 1**
Organization  
(Governance)

**TIER 2**
Mission / Business Process  
(Information and Information Flows)

**TIER 3**
Information System  
(Environment of Operation)

**TACTICAL (OPERATIONAL) RISK FOCUS**

Communicating and sharing risk-related information from the **tactical to strategic level**, that is from the **operators to the executives**.
Current NIST Family

Unified Information Security Framework
Your Toolbox

- NIST Special Publication 800-39
  Managing Information Security Risk: Organization, Mission, and Information System View

- NIST Special Publication 800-30
  Guide for Conducting Risk Assessments

- NIST Special Publication 800-37
  Applying the Risk Management Framework to Federal Information Systems

- NIST Special Publication 800-53
  Security and Privacy Controls for Federal Information Systems and Organizations

- NIST Special Publication 800-53A
  Guide for Assessing the Security Controls in Federal Information Systems and Organizations
Approach: Draft SP 800-161, Supply Chain Risk Management for Federal Information Systems and Organizations

- Building on existing NIST Guidance
- Ability to Implement and Assess
- System Development Life Cycle
- Threat Scenarios & Framework
- ICT SCRM Plan
Expected Changes in 2\textsuperscript{nd} Public Draft

- Rewriting Chapter 1: delineating scope, highlighting specific policy-oriented issues (cost considerations, etc)
- Chapter 3: Delineating “Optional” Controls
- Chapter 3: Control \textit{raison d’etre} for each control
- Appendix G (\textit{new}): ICT SCRM Plan Template
NIST SP 800-161 SCOPE
ICT Supply Chain Infrastructure: Internal Development, Systems, Services, Components

Laws, Regulations, Policies, and other Requirements
Graphic title: ICT SCRM Security Controls in NIST SP 800-161 Section 3.5

OVERLAY

Extract NIST SP 800-53 Rev 4 Security Controls Relevant to ICT SCRM

Additional Information Needed?

NO

Control Title and Tiers Raison d’etre

YES

SECTION 3.5 SHOWS

Control Title, Tiers, and Supplemental Guidance

Control family description, individual controls titles and descriptions, and supplemental guidance

ENHANCED OVERLAY

Add Supplemental Guidance

Add New Controls
Appendix G: ICT SCRM Plan Template

1. Introduction (purpose, scope, authority, audience)
2. Roles/Responsibilities
3. ICT SCRM Controls (Tiers 1, 2, 3)
   • Title
   • How the ICT SCRM control is being implemented or planned to be implemented
   • Applicable scoping guidance
   • Tailoring decisions with justifications.
   • monitoring
Appendix G: ICT SCRM Plan Template

4. Using and Revising ICT SCRM Plan
   4.1 Communicating ICT SCRM Plan
   4.2 Revision and Improvement
      • Triggers: New contracts, events, failures, technologies, etc.
   4.3 Implementing and Assessing Effectiveness
      4.3.1 using and revision SCRM plan
      4.3.2
# SCRM Plan Flow Chart (Acquisition)

<table>
<thead>
<tr>
<th>Title: SCRM Plan – Acquisition Activities/Acquisition Lifecycle</th>
</tr>
</thead>
</table>

- **Tier 1 SCRM Plan**
  - Tier 1 & 2 (Govt.): Tier 2 SCRM Plan
  - Compliance check

- **Tier 3 (Govt.)**
  - Enterprise and end-user risk determination
  - Outline Tier 3 SCRM Plan
  - Seek industry input on SCRM controls using RFI to industry
  - Determine what is needed from bidders
  - Publish SCRM Plan requirements in RFP/RFQ
  - Evaluate and Select
  - Accept final input and build into Govt. plan
  - Independent verification of SCRM plan compliance

- **Market Participants (Suppliers, Service Providers, Integrators)**
  - Respond with recommended SCRM controls
  - High-level plan that satisfies selection criteria
  - Final SCRM plan (which may include sensitive company info)
  - Execute contract IAW SCRM plan

**Contract Award**
SCRM Plan Flow Chart (Lifecycle)

Title: SCRM Plan – System Development Lifecycle (NIST & DHS)

Nominal System Development Lifecycle (NST)

- Initiation
- Development and Acquisition
- Implementation/Assessment
- Operation/Maintenance
- Disposal

Government (Tier 3) SCRM Plan Activities

- Communicate Draft Plan
- Outline Tier 3 (Draft) SCRM Plan
- Perform Acquisition SCRM Plan tasks
- Conduct SCRM Plan Training
- POA&M to address SCRM Plan Weakness
- SCRM Plan Update
- Contingency/Emergency Response
- SCRM Plan Update
- Conduct SCRM Plan Training

System Engineering Lifecycle (DHS)

- Solution Engineering
- Planning
- Requirement Definition
- Design
- Development
- Integration/Test
- Implementation
- Operations/Maintenance
- Disposal
Achieving Balance

- More prescriptive language?
  - or more optional controls and greater system owner autonomy?

- Subject all sensitive systems to SCRM controls?
  - or just procurements that meet criticality threshold conditions?

- Impose risk mitigation even if it drives up costs?
  - or allow programs to manage individually?

- NIST-based SCRM risk paradigm (Tier 1, 2, & 3)?
  - or reliance on system security plans that may have weak SCRM?
Assurance relative to Trust

Managing Effects of Unintentional Defects in Component or System Integrity

Managing Consequences of Unintentional Defects

Managing Consequences of Intentional Exploitable Constructs

TRUST

Quality

Safety

Security
Challenges in Mitigating Risks Attributable to Exploitable ICT/Software & Supply Chains

Several needs arise:

- Need internationally recognized standards to support security automation and processes to provide transparency for informed decision-making in mitigating enterprise risks.
- Need comprehensive diagnostic capabilities to provide sufficient evidence that “code behavior” can be understood to not possess exploitable or malicious constructs.
- Need ‘Assurance’ to be explicitly addressed in standards & capability benchmarking models for organizations involved with security/safety-critical applications.
- Need rating schemes for ICT/software products and supplier capabilities.
Mitigating Risks Attributable to Exploitable Software and Supply Chains

Enterprises seek comprehensive capabilities to:

- Avoid accepting software with **MALWARE** pre-installed.

- 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.

- Determine that exploitable software **WEAKNESSES** that put the users most at risk are mitigated prior to operational acceptance or after put into use.
SSCA Focus on Tainted Components

*Mitigating exploitable risks attributable to non-conforming constructs in ICT*

“Tainted” products are those that are corrupted with malware, or exploitable weaknesses & vulnerabilities that put users at risk

- Enable ‘scalable’ detection and reporting of tainted ICT components
- Leverage/mature related existing standardization efforts
- Provide Taxonomies, schema & structured representations with defined observables & indicators for conveying information:
  - Tainted constructs:
    - Malicious logic/malware (MAEC),
    - Exploitable Weaknesses (CWE);
    - Vulnerabilities (CVE)
  - Attack Patterns (CAPEC)
- Catalogue Diagnostic Methods, Controls, Countermeasures, & Mitigation Practices

Components can become tainted intentionally or unintentionally throughout the supply chain, SDLC, and in Ops & sustainment

*Text demonstrates examples of overlap*
Assurance: Mitigating Attacks That Impact Operations

* Controls include architecture choices, design choices, added security functions, activities & processes, physical decomposition choices, code assessments, design reviews, dynamic testing, and pen testing.
Assurance on the Management of Weaknesses

- Manage Risk During Development
- Assess Deployment Risk
- Operational Mitigation
- Validate/Verify

Diagram:

1. Threat
2. Threat Vector
3. Vulnerability
4. Weakness
5. Control
6. Implementation
7. Test
8. Alarm for Attack/Exploit
9. Block from Attack
10. Eliminate
11. Mitigate

Attack Surface Analysis/Threat Modeling
Many Capabilities Support the Mission

- Hardware
- Capability
- Software
- Systems
- People
- Capability
- Supply Chain Activities
Exploitable Weakness #1 (a vulnerability)

Exploitable Weakness #2 (a vulnerability)

Weakness #3

Mission

Impact from Weakness #1

Impact from Weakness #3
Assurance on the Management of Weaknesses

- **CWE**
- **CVE**
- **CAPEC**

Diagram:
- **Eliminate**
- **Mitigate**
- **Block from Attack**
- **Alarm for Attack/Exploit**

Components:
- Threat
- Threat Vector
- Weakness
- Vulnerability
- Control
- Implementation
- Test
Security Automation “Pipework”

CVE – enabling reporting and patching of vulnerabilities

CWE – identifying and mitigating root cause exploitable weaknesses

CybOX – cyber observables and supply chain exploit indicators

CAPEC – schema attack patterns and software exploits
“Making Security Measureable”: measurablesecurity.mitre.org

Sponsored by DHS with MITRE as technical lead

Resources provided for voluntary adoption

Open, community efforts that are free to use

XML-based

Some important things to note
Leverage Common Weakness Enumeration (CWE) to mitigate risks to mission/business domains

CWE is a formal list of software weakness types created to:
• Serve as a common language for describing software security weaknesses in architecture, design, or code.
• Serve as a standard measuring stick for software security tools targeting these weaknesses.
• Provide a common baseline standard for weakness identification, mitigation, and prevention efforts.

Some Common Types of Software Weaknesses:
Buffer Overflows, Format Strings, Etc. Errors
Structure and Validity Problems Authentication Errors
Common Special Element Manipulations Resource Management Errors
Channel and Path Errors Insufficient Verification of Data
Handler Errors Code Evaluation and Injection
User Interface Errors Randomness and Predictability
Pathname Traversal and Equivalence

cwe.mitre.org
“Weaknesses” address “observables” that might be exploited:

A *weakness* is a property of software, hardware, system (including design and architecture), or process/practice or behavior that, under right conditions, may permit unintended/unauthorized action.

*Common Weakness Enumeration (CWE) v3 http://cwe.mitre.org/

There are many examples of weaknesses:

A weakness is a development or manufacturing mistake, error, bug, flaw, defect, fault, anomaly, or the lack of control in software, firmware, hardware, supply chain process, or operational practice or behavior that could be *exploited*. 
CVE/CWE/CAPEC use in US Government programs

4. VULNERABILITY AND WEAKNESS MANAGEMENT

Purpose and Use

Major attack vectors in US Government programs are commonly identified in National Institute of Standards and Technology (NIST) publications. The Common Vulnerability and Exposures (CVE) database is a key resource for tracking vulnerabilities and associated weaknesses. The Common Weakness Enumeration (CWE) is a comprehensive list of software weaknesses that are used to guide the development of countermeasures. The Common Attack Pattern Enumeration and Classification (CAPEC) identifies attack patterns that can be used to assess the risk of successful attacks.

FY 2013
Chief Information Officer
Federal Information Security Management Act

Reporting Metrics

Table 8 – Responses to Question 4.3

<table>
<thead>
<tr>
<th>Impact Level</th>
<th>For systems in development and/or maintenance:</th>
<th>For systems in production:</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Use methods described in Table 9 to identify and fix instances of common weaknesses, prior to placing that version of the code into production.</td>
<td>Report on configuration and vulnerability levels for hardware assets supporting those systems, giving application owners an assessment of risk inherited from the general support system (network).</td>
</tr>
<tr>
<td>Moderate</td>
<td>Can the organization find SCAP compliant tools and good SCAP content?</td>
<td>Can the organization find SCAP compliant tools and good SCAP content?</td>
</tr>
<tr>
<td>Low</td>
<td>Use methods described in Table 9 to identify and fix instances of common weaknesses, prior to placing that version of the code into production.</td>
<td></td>
</tr>
</tbody>
</table>

Table 9 – Methods to Identify and Fix Instances of Common Weaknesses

<table>
<thead>
<tr>
<th>Identify Universe Enumeration</th>
<th>Find Instances Tools and Languages</th>
<th>Assess Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Common Weakness Enumeration (CWE)</td>
<td>• Static Code Analysis tools</td>
<td>• Common Weakness Scoring System (CWSS)</td>
</tr>
<tr>
<td>• Web scanners for web-based applications</td>
<td>• Manual code reviews (especially for weaknesses not covered by the automated tools)</td>
<td></td>
</tr>
<tr>
<td>• Common Attack Pattern Enumeration and Classification (CAPEC)</td>
<td>• Dynamic Code Analysis tools</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Web scanners for web-based applications</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• PEN testing for attack types not covered by the automated tools.</td>
<td></td>
</tr>
</tbody>
</table>

See guidance that describes the purpose and use of these tools and how they can be used today in a practical way to improve security of software during development and maintenance.

Prepared by:
US Department of Homeland Security
Office of Cybersecurity and Communications
Federal Network Resilience

November 30, 2012
Software Assurance.—The term “software assurance” means 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. Sect 933

Example Use:

DoD Software-based System

Program Office Milestone Reviews with OSD on SwA

Program Protection Plan’s “Application of Software Assurance Countermeasures”

Development Process
- Static Analysis
- Design Inspection
- Code Inspections
- CVE
- CAPEC
- CWE
- Pen Test
- Test Coverage

Operational System
- Failover Multiple Supplier Redundancy
- Fault Isolation
- Least Privilege
- System Element Isolation
- Input checking/validation
- SW load key

Development Environment
- Source
- Release Testing
- Generated code inspection
DoD Program Protection Plan (PPP) Software Assurance Methods

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

Additional Guidance in PPP Outline and Guidance
13.7.3. Software Assurance
13.7.3.1. Development Process
13.7.3.1.1 Static Analysis
13.7.3.1.2 Design Inspection
13.7.3.1.3 Code Inspection
13.7.3.1.4. Common Vulnerabilities and Exposures (CVE)
13.7.3.1.5. Common Attack Pattern Enumeration and Classification (CAPEC)
13.7.3.1.6. Common Weakness Enumeration information (CWE)
13.7.3.1.7. Penetration Test
13.7.3.1.8. Test Coverage
13.7.3.2. Operational System
13.7.3.2.1. Failover Multiple Supplier Redundancy
13.7.3.2.2. Fault Isolation
13.7.3.2.3. Least Privilege
13.7.3.2.4. System Element Isolation
13.7.3.2.5. Input Checking/Validation
13.7.3.2.6. Software Encryption and Anti-Tamper Techniques (SW load key)
13.7.3.3. Development Environment
13.7.3.3.1. Source Code Availability
13.7.3.3.2. Release Testing
13.7.3.3.3. Generated Code Inspection
13.7.3.3.3. Additional Countermeasures
The paper also contains two important, additional sections for each listed practice that will further increase its value to implementers—Common Weakness Enumeration (CWE) references and Verification guidance.

Example:

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

<table>
<thead>
<tr>
<th>Threat Identified</th>
<th>Design Element(s)</th>
<th>Mitigation</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session hijacking</td>
<td>GUI</td>
<td>Ensure random session identifiers of appropriate length</td>
<td>Collect session identifiers over a number of sessions and examine distribution and length</td>
</tr>
<tr>
<td>Tampering with data in transit</td>
<td>Process A on server to Process B on client</td>
<td>Use SSL to ensure that data isn’t modified in transit</td>
<td>Assert that communication cannot be established without the use of SSL</td>
</tr>
</tbody>
</table>

CWE References

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

- CWE-207: Improper authentication is an example of a weakness that could be exploited by an attacker threat.
- CWE-266: Permissions, Privileges, and Access Controls is a parent weakness of many Tampering, Repudiation, and Repudiation of Privilege threats.
- CWE-311: Missing encryption of Sensitive Data is an example of an Information Disclosure threat.
- CWE-400: Uncontrolled resource consumption is an example of an unmitigated Denial of Service threat.
## Practical Security Stories and Security Tasks for Agile Development Environments

### Industry Uptake - Agile

**Table of Contents**
- Problem Statement and Target Audience
- Overview
- Assumptions
- Section 1: Agile Development Methodologies and Security
- How to Choose the Security-focused Stories and Security Tasks?
- Story and Task Prioritization Using "Security Debt"
- Residual Risk Acceptance
- Section 2a: Security-focused Stories and Associated Security Tasks
- Section 2b: Operational Security Tasks
- Section 3: Tasks Requiring the Help of Security Experts
- Appendix A: Residual Risk Acceptance
- Glossary
- References
- About SAFECode

---

<table>
<thead>
<tr>
<th>No.</th>
<th>Security-focused story</th>
<th>Backlog task(s)</th>
<th>SAFECode Fundamental Practice(s)</th>
<th>CWE-ID</th>
</tr>
</thead>
</table>
| 1   | As a(n) architect/developer, I want to ensure **AND** as QA, I want to verify allocation of resources within limits or throttling | **[A]** Clearly identify resources. A few examples:  
- Number of simultaneous connections to an application on a web server from same user or from different users  
- File size that can be uploaded  
- Maximum number of files that can be uploaded to a file system folder  
**[A/D]** Define limits on resource allocation.  
**[T]** Conduct performance/stress testing to ensure that the numbers chosen are realistic (i.e. backed by data).  
**[A/D/T]** Define and test system behavior for correctness when limits are exceeded. A few examples:  
- Rejecting new connection requests  
- Preventing simultaneous connection requests from the same user/IP, etc.  
- Preventing users from uploading files greater than a specific size, e.g., 2 MB  
- Archiving data in file upload folder when a specific limit is reached to prevent file system exhaustion | - Validate Input and Output to Mitigate Common Vulnerabilities  
- Perform Fuzz/Robustness Testing | CWE-770 |
Idaho National Labs SCADA Report

NSTB Assessments
Summary Report:
Common Industrial Control
System Cyber Security
Weaknesses

May 2010

SECURE CONTROL SYSTEM/ENTERPRISE ARCHITECTURE

Level 4
Enterprise Systems: Business Planning
and Logistics / Engineering Systems

Corporate Network

Level 3
Operations Management: System Management / Supervisory Control

LAN / WAN / DMZ

ICS Business Application Client
ICS Web Application Client
Corporate Hosts
ICCP Server
OPC Server
Information Server
Application Server

Replicated Database
Web Server

Level 2
Supervisory Control Equipment: Supervisory Control Functions / Site Monitoring and Local Display

Supervisory Control LAN

Supervisory Control
Local Display
Real-time Database
Communications Processor

Level 1
Control Equipment: Protection and Local Control Devices

Control Network

RTU
Distributed Control
PLC

Level 0
Equipment Under Control: Sensors and Actuators

I/O Network

Temperature Sensor
Pressure Sensor
Relay
Table 27. Most common programming errors found in ICS code.

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

**Application Security Risks**

- **Threat Agents:**
  - **Attack Vectors:**
    - **Security Weakness:**
      - **Detectability:**
        - **Impact:**
          - **Application Specific**
            - **Exploitability:**
              - **Prevalence:**
                - **detectability:**
                  - **Impact:**
                    - **Application Impacts**
                      - **Business Impacts**
                        - **Application Specific Exploitability**
                          - **Prevalence**
                            - **Detectability**
                              - **Impact**
                                - **Application Impacts**
                                  - **Business Impacts**
                                    - **Application Specific Exploitability**
                                      - **Prevalence**
                                        - **Detectability**
                                          - **Impact**
                                            - **Application Impacts**
                                              - **Business Impacts**
                                                - **Application Specific Exploitability**
                                                    - **Prevalence**
                                                      - **Detectability**
                                                        - **Impact**
                                                          - **Application Impacts**
                                                            - **Business Impacts**
                                                              - **Application Specific Exploitability**
                                                                - **Prevalence**
                                                                  - **Detectability**
                                                                    - **Impact**
                                                                      - **Application Impacts**
                                                                        - **Business Impacts**

### How Do I Prevent Injection?

The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. For SQL calls, this means using bind variables in all prepared statements and stored procedures, and avoiding dynamic queries.

Checking the code is a fast and accurate way to see if the application uses interpreters safely. Code analysis tools can help a security analyst find the use of interpreters and trace the data flow through the application. Penetration testers can validate these issues by crafting exploits that confirm the vulnerability.

Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection flaws exist. Scanners cannot always reach interpreters and have difficulty detecting whether an attack was successful. Poor error handling makes injection flaws easier to discover.

### Example Attack Scenarios

**Scenario #1:** The application uses untrusted data in the construction of the following vulnerable SQL call:

```java
String query = "SELECT * FROM accounts WHERE custId" + request.getParameter("id") + "=42;"
```

**Scenario #2:** Similarly, an application’s blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```java
Query hqlQuery = session.createQuery("FROM accounts WHERE custId" + request.getParameter("id") + "=42;"
```

In both cases, the attacker modifies the 'id' parameter value in her browser to send: '1" OR '1'="' or '1=1'. For example:

```java
http://example.com/app/accountView?id=1' OR '1=1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.

### References

**OWASP**

- OWASP SQL Injection Prevention Cheat Sheet
- OWASP Query Parameterization Cheat Sheet
- OWASP Command Injection Article
- OWASP XML eXtensible Entity (XXE) Reference Article
- ASVS: Output Encoding/Escaping Requirements (V6)

**External**

- CWE Entry 77 on Command Injection
- CWE Entry 89 on SQL Injection
- CWE Entry 564 on Hibernate Injection
Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses

• Identifies mission/business risks attributable to the respective weaknesses; identifies common attacks that exploit those weaknesses, and provides recommended practices for preventing the weaknesses.

  – CWE focuses on stopping vulnerabilities at the source by educating designers, programmers, and QA/testers on how to eliminate all too-common mistakes before software is even shipped.

  – CWE Top-N lists serve as tools for education, training and awareness to help programmers prevent the kinds of vulnerabilities that plague the software industry.

  – Software consumers could use the same list to help them to ask for more secure software.

  – Software managers and CIOs can use the CWE list as a measuring stick of progress in their efforts to secure their 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](http://cwe.mitre.org) and [http://capec.mitre.org](http://capec.mitre.org).

<table>
<thead>
<tr>
<th>CWE</th>
<th>Related Attack Pattern</th>
<th>Mission/Business Risks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CAPEC-64: Using Slashes and URL Encoding Combined to Bypass Validation Logic</td>
<td>Execute unauthorized code or commands</td>
</tr>
<tr>
<td></td>
<td>CAPEC-76: Manipulating Input to File System Calls</td>
<td>Modify files or directories</td>
</tr>
<tr>
<td></td>
<td>CAPEC-78: Using Escaped Slashes in Alternate Encoding</td>
<td>Read files or directories</td>
</tr>
<tr>
<td></td>
<td>CAPEC-79: Using Slashes in Alternate Encoding</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-139: Relative Path Traversal</td>
<td></td>
</tr>
<tr>
<td>CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>CAPEC-6: TCP Header</td>
<td>DoS: crash / exit / restart</td>
</tr>
<tr>
<td></td>
<td>CAPEC-15: Command Delimiters</td>
<td>Execute unauthorized code or commands</td>
</tr>
<tr>
<td></td>
<td>CAPEC-43: Exploiting Multiple Input Interpretation Layers</td>
<td>Hide activities</td>
</tr>
<tr>
<td></td>
<td>CAPEC-88: OS Command Injection</td>
<td>Modify application data</td>
</tr>
<tr>
<td></td>
<td>CAPEC-108: Command Line Execution through SQL Injection</td>
<td>Modify application data</td>
</tr>
<tr>
<td>CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>CAPEC-18: Embedding Scripts in Nonscript Elements</td>
<td>Bypass protection mechanism</td>
</tr>
<tr>
<td></td>
<td>CAPEC-19: Embedding Scripts within Scripts</td>
<td>Execute unauthorized code or commands</td>
</tr>
<tr>
<td></td>
<td>CAPEC-32: Embedding Scripts in HTTP Query Strings</td>
<td>Read application data</td>
</tr>
<tr>
<td></td>
<td>CAPEC-63: Simple Script Injection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-85: Client Network Footprinting (using AJAX/XSS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-86: Embedding Script (XSS) in HTTP Headers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-91: XSS in IMG Tags</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-106: Cross Site Scripting through Log Files</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-198: Cross-Site Scripting in Error Pages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-199: Cross-Site Scripting Using Alternate Syntax</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-209: Cross-Site Scripting Using MIME Type Mismatch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-232: Exploitation of Privilege Trust</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-243: Cross-Site Scripting in Attributes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-244: Cross-Site Scripting via Encoded URI Schemes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-245: Cross-Site Scripting Using Doubled Characters, e.g. <code>%3C%3E</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-246: Cross-Site Scripting Using Flash</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAPEC-247: Cross-Site Scripting with Masking through Invalid Characters in Identifiers</td>
<td></td>
</tr>
</tbody>
</table>
Prioritizing weaknesses to be mitigated

OWASP Top 10

CWE/SANS Top 25

Lists are a good start but they are designed to be broadly applicable

We would like a way to specify priorities based on business/mission risk
Prioritizing Technical Impacts: CWE’s Common Consequences

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’)

Applicable Platforms

Languages

Technology Classes

Database-Server

Modes of Introduction

This weakness typically appears in data-rich applications that save user inputs in a database.

Common Consequences

Scope

Effect

Confidentiality

Technical Impact: Read application data

Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.

Access Control

Technical Impact: Bypass protection mechanism

If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.

Access Control

Technical Impact: Bypass protection mechanism

If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability.

Integrity

Technical Impact: Modify application data

Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.
Common Weakness Risk Analysis Framework (CWRAF)

How do I **identify** which of the 900+ CWE’s are most important for my specific business domain, technologies and environment?

Common Weakness Scoring System (CWSS)

How do I **rank** the CWE’s I care about according to my specific business domain, technologies and environment?

How do I identify and score weaknesses important to my organization?
CWE’s all lead to these Technical Impacts

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
Utilizing a Priority List of Weaknesses

Which static analysis tools and Pen Testing services find the CWEs I care about?

- Code Review
- Static Analysis Tool A
- Static Analysis Tool B
- Pen Testing Services

CWEs a capability claims to cover

Most Important Weaknesses (CWEs)

Which static analysis tools and Pen Testing services find the CWEs I care about?
Take Advantage of the Multiple Detection Methods

- Different assessment methods are effective at finding different types of weaknesses
- Some are good at finding the cause and some at finding the effect

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Static Code Analysis</th>
<th>Penetration Test</th>
<th>Data Security Analysis</th>
<th>Code Review</th>
<th>Architecture Risk Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-Site Scripting (XSS)</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL Injection</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insufficient Authorization Controls</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Broken Authentication and Session Management</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Information Leakage</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improper Error Handling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insecure Use of Cryptography</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cross Site Request Forgery (CSRF)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Denial of Service</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Poor Coding Practices</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Technical Impacts – Common Consequences

Detection Methods

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

Detection Methods

**Automated Static Analysis**

This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives. Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or do not require any code changes. Automated static analysis might not be able to detect the usage of custom API functions or third-party libraries that indirectly invoke SQL commands, leading to false negatives - especially if the API/library code is not available for analysis.

This is not a perfect solution, since 100% accuracy and coverage are not feasible.

**Automated Dynamic Analysis**

This weakness can be detected using dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

**Effectiveness: Moderate**

**Manual Analysis**

Manual analysis can be useful for finding this weakness, but it might not achieve desired code coverage within limited time constraints. This becomes difficult for weaknesses that must be considered for all inputs, since the attack surface can be too large.

**Demonstrative Examples**

**Example 1**

In 2000, a large number of web servers were compromised using the same SQL injection attack string. This single...
Detection Methods

The "Detection Methods" field within many CWE entries conveys information about what types of assessment activities that weakness can be found by. Increasing numbers of CWE entries will have this field filled in over time. The recent Institute of Defense Analysis (IDA) State of the Art Research report conducted for DoD provides additional information for use across CWE in this area. Labels for the Detection Methods being used within CWE are:

- Automated Analysis
- Automated Dynamic Analysis
- Automated Static Analysis
- Black Box
- Fuzzing
- Manual Analysis
- Manual Dynamic Analysis
- Manual Static Analysis
- White Box

With this type of information (shown in the table below), we can see which of the specific CWEs that can lead to a specific type of technical impact are detectable by dynamic analysis, static analysis, and fuzzing evidence and which ones are not.

This table is incomplete, because many CWE entries do not have a detection method listed.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute unauthorized code or commands</td>
<td>78, 120, 129, 131, 476, 805</td>
<td>78, 79, 98, 120, 129, 131, 134, 190, 426, 798, 805</td>
<td>79, 129</td>
<td>98, 120, 131, 190, 426, 494, 494, 805</td>
<td>476, 798</td>
<td>78, 798</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modify data</td>
<td>311, 327</td>
<td>78, 89, 129, 131, 134, 135, 426, 798</td>
<td>129, 190, 319</td>
<td>89, 131, 190, 311, 78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

New Detection Methods Launched Feb 17
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute unauthorized code or commands</td>
<td>78, 120, 129, 131, 476, 805</td>
<td>78, 79, 98, 120, 129, 131, 134, 190, 426, 798, 805</td>
<td>79, 129, 134, 190, 426, 494, 698, 798</td>
<td>98, 120, 131, 190, 426, 494, 805</td>
<td>476, 798</td>
<td>78, 798</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modify data</td>
<td>311, 327</td>
<td>78, 89, 129, 131</td>
<td>78, 89, 129, 131, 190, 352</td>
<td>129, 190, 319</td>
<td>89, 131, 190, 311, 327, 352</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DoS: unreliable execution</td>
<td>78, 120, 129, 131, 400, 476, 665, 805</td>
<td>78, 120, 129, 131, 190, 352, 400, 426, 805</td>
<td>129, 190, 426, 690</td>
<td>120, 131, 190, 352, 426, 805</td>
<td>476, 665</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DoS: resource consumption</td>
<td>120, 400, 404, 770, 805</td>
<td>120, 190, 400, 770, 805</td>
<td>190</td>
<td>400, 770</td>
<td>120, 190, 805</td>
<td>404</td>
<td>770</td>
<td>412</td>
<td></td>
</tr>
<tr>
<td>Bypass protection mechanism</td>
<td>89, 400, 601, 665</td>
<td>79, 89, 190, 352, 400, 601, 798</td>
<td>14, 79, 184, 190, 733, 798</td>
<td>89, 190, 352</td>
<td>665, 798</td>
<td>601, 798, 807</td>
<td>14, 733</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hide activities</td>
<td>327</td>
<td>78</td>
<td>78</td>
<td></td>
<td>327</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CWRAF/CWSS Provides Risk Prioritization for CWE throughout Software Life Cycle

- Enables education and training to provide specific practices for eliminating software fault patterns;
- Enables developers to mitigate top risks attributable to exploitable software;
- Enables testing organizations to use suite of test tools & methods (with CWE Coverage Claims Representation) that cover applicable concerns;
- Enables users and operation organizations to deploy and use software that is more resilient and secure;
- Enables procurement organizations to specify software security expectations through acquisition of software, hosted applications and services.
When should I focus on Weaknesses and Vulnerabilities?

- **Focus on Weaknesses**: A type of defect that *may be* exploitable.
  - **Keep Weaknesses from becoming vulnerabilities**
- **Focus on Vulnerabilities**: Something in code that *can actually* be exploited.
Automation can help...

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

**Verification**
- 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)

**Deployment**
- Common Vulnerabilities and Exposures (CVE)
- Open Vulnerability Assessment Language (OVAL)
- Malware Attribute Enumeration and Characterization (MAEC)
- Cyber Observables eXpression (CybOX)
# Enable Automation: Software Assurance/ SCRM

## Purpose/Objective:
Gain confidence that software is free from exploitable vulnerabilities either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner through the use of automation and expressions of software attributes.

- CVE – Common Vulnerability
- CWE – Common Weakness Enumeration
- CAPEC – Common Attack Patterns
- OVAL – Open Vulnerability Assessment Language
- MAEC – Malware Attribute Enumeration Characterization
- CybOX – Cyber Observable eXpression (common shared-schema of MAEC, CAPEC, & STIX)

## Accomplishments to Date:
- 93 organizations with 158 products & services are CVE compatible - 61,000+ usable identifiers providing information to the global IT community for threat alerts and patch management;
- 49 CWE v2 compatible products & services from 31 organizations used by thousands of development organizations to mitigate exploitable constructs in design & code - 900+ CWE exploitable constructs to avoid identified risks which lead to better software;
- CAPEC attack patterns provide meaningful test cases leading to resilience of products and processes;
- OVAL v5.10 tools embedded in 33 products/services;
- MAEC enables shared info exchange & analysis.

## Upcoming Activities/Next Steps:
- Collaborate in releasing CWE v3 and CAPEC v3
- Enhance use of DHS-sponsored security automation languages & enumerations by supporting emerging standards; Collaborate with industry on SDLC processes that leverage security automation
- Collaborate on SwA/SCRM planning & training material

## Potential Risks/Constraints:
- Software development processes are fueled by demand for functionality and low cost rather than security and resilience to exploitation.
- Use of security automation not fully realized: Not all CVEs are patched; 50% of CWEs have no associated CVEs because many discovered weaknesses are unreported; represents greater risk of zero-day attacks

## What do others need to know?
Software Assurance Curriculum Project

- **Vol I: Master of Software Assurance Reference Curriculum**

In Dec 2010 the IEEE Computer Society and the ACM recognized the Master of Software Assurance (MSwA) Reference Curriculum as a certified master’s degree program in SwA—the first curriculum to focus on assuring the functionality, dependability, and security of software and systems.

- **Vol II: SwA Undergraduate Course Outlines**

see [www.sei.cmu.edu/library/abstracts/reports/10tr019.cfm](http://www.sei.cmu.edu/library/abstracts/reports/10tr019.cfm) to download the PDF version of the report CMU/SEI-2010-TR-019

- **Vol III: Master of SwA Course Syllabi**

- **Vol IV: Community College Education**

  - Report on “Integrating the MSwA Reference Curriculum into Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems” provides reference and guidance material.

  - To facilitate implementation, the MSwA project team is offering assistance, free of charge, to educational institutions looking to launch an MSwA degree program.

  - For more information, go to [https://buildsecurityin.us-cert.gov/bsi/1165-BSI.html](https://buildsecurityin.us-cert.gov/bsi/1165-BSI.html).
Developed to support the following uses:

• Provide employers of SwA personnel with a means to assess the SwA capabilities of current and potential employees.

• Offer guidance to academic or training organizations:
  – develop SwA courses to support the needs of organizations that are hiring and developing SwA professionals.
  – Enhance SwA curricula guidance by providing information about industry needs and expectations for competent SwA professionals.

• Provide direction and a progression for the development and career planning of SwA professionals.

• Provide support for professional certification and licensing.
SwA in Acquisition & Outsourcing
• Software Assurance in Acquisition and Contract Language
• Software Supply Chain Risk Management and Due-Diligence

SwA in Development *
• Risk-based Software Security Testing
• Requirements and Analysis for Secure Software
• Architecture and Design Considerations for Secure Software
• Secure Coding and Software Construction
• Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses
* All include questions to ask developers

SwA Life Cycle Support
• SwA in Education, Training and Certification

SwA Pocket Guides and SwA-related documents are collaboratively developed with peer review; they are subject to update and are freely available for download via the DHS Software Assurance Community Resources and Information Clearinghouse at https://buildsecurityin.us-cert.gov/swa (see SwA Resources)
Security-Enhanced Process Improvements

Organizations that provide security engineering & risk-based analysis throughout the lifecycle will have more resilient software products / systems.

“Build Security In” throughout the lifecycle

- Leverage Software Assurance resources (freely available) to incorporate in training & awareness
- Modify SDLC to incorporate security processes and tools (should be done in phases by practitioners to determine best integration points)
- Avoid drastic changes to existing development environment and allow for time to change culture and processes
- Make the business case and balance the benefits
- Retain upper management sponsorship and commitment to producing secure software.

* Adopted in part from “Software Assurance: Mitigating Supply Chain Risks” (DHS NCSD SwA); “What to Test from a Security Perspective for the QA Professional” (Cigital) and “Neutralizing the Threat: A Case Study in Enterprise-wide Application Security Deployments” (HP Fortify Software & Accenture Security Technology Consulting).
DHS Software & Supply Chain Assurance Outreach

- Co-sponsor SSCA Forum & WGs for government, academia, and industry to facilitate ongoing public-private collaboration.
- Provide SwA presentations, workshops, and tracks at conferences.
- Co-sponsor issues of CROSSTALK to “spread the word”
  - Sep/Oct 2009 issue on “Resilient Software”
  - Mar/Apr 2010 issue on “System Assurance”
  - Sep/Oct 2010 issue on “Game Changing Tools & Practices”
  - Mar/Apr 2011 issue on “Rugged Software”
  - Sep/Oct 2011 issue on “Protecting against Predatory Practices”
  - Mar/Apr 2012 issue on “Securing a Mobile World”
  - Sep/Oct 2012 issue on “Resilient Cyber Ecosystem”
  - Mar/Apr 2013 issue on “Supply Chain Risk Management”
  - Sep/Oct 2013 issue on “Securing the Cloud”
  - Mar/Apr 2014 issue on “Mitigating Risks from Counterfeit & Tainted Products”
- Collaborate with standards organizations, consortiums, professional societies, education/training initiatives in promoting SwA.
- Provide free SwA resources via “BuildSecurityIn” website to promote secure development methodologies (since Oct 05).
- Host SSCA Community Resources & Information Clearinghouse via https://buildsecurityin.us-cert.gov/SwA.
Mitigating Risks of Counterfeit and Tainted Components

Non-Malicious Taint
Bad Hygiene is as Dangerous to the Mission as Malicious Intent

Robert A. Martin, MITRE Corporation

Abstract. Success of the mission should be the focus of software and supply chain assurance activities regardless of what activity produces the risk. It does not matter if a malicious behavior is the cause. It does not matter if it is malicious logic inserted at the factory or inserted through an update after fielding. It does not matter if it comes from an inescapable bug or from a failure to understand how an attacker could exploit a software feature. Issues from bad software hygiene, like inadvertent coding flaws or weak architectural constructs are as dangerous to the mission as malicious acts. Even innocuous errors can put high hygiene and quality in the medical and food industries at risk. At any juncture of fact, similar types of vulnerabilities need to be applied to software and hardware. Until both malicious and non-malicious aspects of taint can be dealt with in a way that is visible and verifiable, there will be a continued lack of confidence and assurance in delivered capabilities throughout their lifecycle.

Background
Every piece of information and communications technology (ICT) hardware—this includes computers as well as any device that stores, processes, or transmits data—has an initially embedded software component that requires follow-up support and sustainment. The concept of supply chain risk management (SCRM) must be applied to both the software and hardware components within a mission's life cycle. Even if SCRM is maintained, the supply chain for ongoing sustainment support of the software is often disconnected from the support for the hardware (e.g., continued software maintenance contracts with third parties rather than the original manufacturer). As a result, supply chain assurance regarding software requires a slightly unique approach as compared to the hardware electronics.

Some may want to focus on just “how hanging fruit” like banning suspect products by the country that comes from or by the ownership of the producer due to their known nature and ignorance of the cold war. The software aspect of ICT like the exploitatable vulnerabilities outlined in this article. It is a misconception that “adding” software assurance to the mix of supply chain concerns and activities will add too much complexity, thereby making SCRM even harder to perform. Some organizations and sectors are already developing standards of care and due diligence that directly address these unattended and bad hygiene types of issues. That said, such practices for avoiding bad hygiene issues that make software unfair for its intended purposes are not the norm across most of the industries involved in creating and supporting software-based products. Mitigating risk to the mission is a critical objective and including software assurance as a fundamental aspect of SCRM for ICT equipment is a critical component of delivering mission assurance.

During the past several decades, software-based ICT capabilities have become the basis of almost every aspect of today’s commerce, governance, national security, and recreation. Software-based devices are in homes, vehicles, communications, and toys. Unfortunately, software, the basis of these cyber capabilities, can be unpredictable since there are now underlying risks software has to follow as opposed to the rest of our material world which is constrained by the laws of gravity, chemistry, and physics with core factors like Planck’s Constant. This is even more true given the variety and level of skills and training of those who create and evolve cyber capabilities. The result is that for the foreseeable future there will remain a need to address the types of quality and integrity problems that leave software unreliable, attackable, and brittle. This includes addressing the problems that allow malicious and exploitable vulnerabilities to be accidentally inserted into products during development, packaging, or updates due to poor software hygiene practices.

Computer languages specifications are historically vague and loosely written. (Note: ISO/IEC JTC1, SC22 issued a Technical Report [1] with guidance for selecting languages and using languages more secure and reliably) There is often a lack of concern for resilience, robustness, and security in the variety of development tools used to build and deploy software. And there are gaps in the skills and education of those that manage, specify, create, test, and field these software-based products. Additionally, software-based products are available to attackers who share the same problem space and know how to exploit vulnerabilities that their creators never intended. Traditionally this has led to calls for improved security functionality and more rigorous review, testing, and maintenance of the way ICT is created. However, that approach fails to account for the core differences between the engineering of software-based products and other engineering disciplines.

Those differences are detailed in this article.

The need to address those differences has accelerated as more of the nation’s critical industrial, financial, and military capabilities rely on software-based products that comprise the expanding cyber world. ICT systems must be designed to withstand attacks and offer resilience through better security, avoiding of known weaknesses in code, architecture, and design. Additionally, ICT systems should be created with designers-in-protection capabilities to address unforeseen attacks by making them intrinsically more robust and resilient so that there are fewer ways to impact the system. This same concern has been expressed by Congress with the inclusion of a definition of “Software Assurance” in Public Law 112-229 Section 203 [2] where they directed DoD to specifically address software assurance of its systems.
Getting Started in Software Assurance (SwA)

Success of the mission should be the focus of software and other assurance activities. Although increasing automation of various capabilities has provided great boons to our organizations, this automation is also at risk for becoming a targeted focus for attackers’ attentions and techniques. Recognizing that your software and supply chain have exploitable weaknesses is a major step to improving the reliability, resilience, and integrity of your software when it faces attacks.

The key to gaining assurance about your software is to make incremental improvements when you develop it, when you buy it, and when others create it for you. No single remedy will absolve or mitigate all of the weaknesses in your software, or the risk. However, by blending several different methods, tools, and change in culture, one can obtain greater confidence that the important functions of the software will be there when they are needed and the worst types of failures and impacts can be avoided.

There is no crystal ball, or magic wand one can use to ensure software is absolutely secure against the unknown. However, there are ways to limit negative impacts and improve confidence in software-based capabilities and their ability to deliver their part to the organization’s mission.

This section of the CWE Web site introduces specific steps you can take to 1) assess your individual software assurance situation and 2) compose a tailored plan to strengthen assurance of integrity, reliability, and resilience of your software and its supply chain. Learn more by following the links below:

- Engineering for Attacks
- Software Quality
- Prioritizing Weaknesses Based Upon Your Organization’s Mission
- Detection Methods
- Manageable Steps
- Software Assurance Pocket Guide Series
- Staying Informed
- Finding More Information about Software Assurance
Focuses on making security a normal part of software engineering

https://buildsecurityin.us-cert.gov

Provides resources for stakeholders with interests in Software Assurance, Supply Chain Risk Management, and Security Automation

https://buildsecurityin.us-cert.gov/swa
Software & Supply Chain Assurance:

Enabling Enterprise Resilience through Security Automation, Software Assurance, and Supply Chain Risk Management

Joe Jarzombek, PMP, CSSLP
Director for Software & Supply Chain Assurance
Stakeholder Engagement & Cyber Infrastructure Resilience
Cyber Security & Communications
joe.jarzombek@hq.dhs.gov