Software & Supply Chain Assurance:

Enabling Enterprise Resilience through 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

Mitigating Cyber-Physical Risk Exposures Attributable to External Dependencies on ICT Supply Chain Components and Services
Public-Private Collaboration Efforts for Security Automation, Software Assurance, and Supply Chain Risk Management

Next SSCA Forum 9-11 March 2015 at MITRE in McLean, Virginia
DHS Software & Supply Chain Assurance Strategy

Support Response

Enable Automation

Outreach & Collaborate

Policy decisions influence tool design

Tools allow standards and policies to work

Manage Software & Supply Chain Risks

Influence Policy

Influence Standards

Standards are a foundation of good policy

Policy decisions influence tool design

Tools allow standards and policies to work

Standards are implemented by tools

Security Automation, Acquisition Risk Analysis/Modeling

Publications, Websites, FedVTE, Training & Education Working Groups, Forums

US Fed & IT Acquisition


ISO/IEC JTC1 The Open Group ITU-T CYBEX OMG

NIST Spec Pubs SP800-161, 160 & 53

Standards are a foundation of good policy

Forums

CFIUS Review CRR

NPPD Contract Reviews

Policy decisions influence tool design

Tools allow standards and policies to work

Standards are implemented by tools

Security Automation, Acquisition Risk Analysis/Modeling

Publications, Websites, FedVTE, Training & Education Working Groups, Forums

US Fed & IT Acquisition


ISO/IEC JTC1 The Open Group ITU-T CYBEX OMG

NIST Spec Pubs SP800-161, 160 & 53

Standards are a foundation of good policy

Forums
DHS Software & Supply Chain Assurance Outreach

► Co-sponsor SSCA Forum & WGs for government, academia, and industry to facilitate ongoing public-private collaboration.
► 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”
  ▪ Mar/Apr 2015 issue on “Test & Diagnostics”
► 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
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
Interdependencies Between Physical & Cyber Capabilities – Convergence of Safety, Security and Resilience Considerations

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
Exploitable Software Weaknesses (CWEs) are exploit targets/vectors for future Zero-Day Attacks

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)

Security Feature
Known weaknesses plague the security threat landscape

Many of the biggest security risks are issues known about for decades, leaving organizations unnecessarily exposed; organizations must employ fundamental security tactics to address known vulnerabilities to eliminate significant amounts of risk.

- 44% of known breaches come from vulnerabilities that are 2-4 years old.
- Server misconfigurations were the number one vulnerability.
- Additional avenues of attack were introduced via connected devices.
- The primary causes of commonly exploited software vulnerabilities are defects, bugs, and logic flaws.
Software Assurance Addresses Exploitable Software:
Outcomes of non-secure practices and/or malicious intent

Exploitation potential of vulnerability is independent of “intent”

Defects

EXPLOITABLE SOFTWARE

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,
-- Cybersecurity 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 determines ‘fitness for use’ for ‘technically acceptable’ criteria?
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
Reduced Visibility, Understanding and Control
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

Multitiered Organizational Risk Management

Security Controls

Risk Assessment

SP 800-161

SP 800-39

SP 800-53r4

SP 800-30
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
SSCA Focus on Tainted Components

*Mitigating risks attributable to exploitable 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*
Automation is one piece of the SwA & SCRM puzzle.
“Making Security Measureable”: measurablesecurity.mitre.org

Sponsored by DHS CS&C with MITRE as technical lead

Open, community efforts that are free to use

Resources provided for voluntary adoption

XML-based
Software Assurance

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.*

Derived From: CNSSI-4009

Automation

Languages, enumerations, registries, tools, and repositories

throughout the Lifecycle

Including design, coding, testing, deployment, configuration and operation
Cyber Threats Emerged Over Time

1980's
- "stealth"/advanced scanning techniques
- automated probes/scans
- sniffers
- network mgmt. diagnostics
- hijacking sessions
- back doors
- disabling audits
- Internet social engineering attacks
- password cracking
- exploiting known vulnerabilities
- password guessing

1990's
- widespread attacks on DNS infrastructure
- executable code attacks (against browsers)
- automated widespread attacks
- GUI intruder tools
- packet spoofing
- burglaries
- www attacks
- distributed attack tools
- increase in wide-scale Trojan horse distribution
- Windows-based remote controllable Trojans (Back Orifice)

2000's
- email propagation of malicious code
- automated probes/scans
- www attacks
- anti-forensic techniques
- home users targeted
- increase in tailored worms
- sophisticated command & control
- diffuse spyware
- techniques to analyze code for vulnerabilities without source code
- increase in wide-scale Trojan horse distribution
- increase in tailored worms

2010's
- DDoS attacks
- binary encryption
- techniques to analyze code for vulnerabilities without source code
- distributed attack tools
- Windows-based remote controllable Trojans (Back Orifice)
Solutions Also Emerged Over Time

- **1980’s**
  - Password guessing
  - Exploiting known vulnerabilities
  - Burglaries
  - Packet spoofing

- **1990’s**
  - Disabling audits
  - Internet social engineering attacks
  - Sniffers
  - GUI intruder tools
  - Network mgmt. diagnostics
  - Executable code attacks (against browsers)

- **2000’s**
  - Automated widespread attacks
  - Automated probes/scans
  - Email propagation of malicious code
  - DDoS attacks
  - Binary encryption
  - Increase in tailored worms
  - Sophisticated command & control
  - Anti-forensic techniques
  - Home users targeted
  - Increase in wide-scale Trojan horse distribution
  - Windows-based remote controllable Trojans (Back Orifice)

- **2010’s**
  - WWW attacks
  - Diffuse spyware
  - Techniques to analyze code for vulnerabilities without source code
  - Increase in wide-scale Trojan horse distribution
  - Sophisticated command & control
Architecting Security with Information Standards for COIs
Operational Enterprise Networks

Development & Sustainment

Security Management Processes

Enterprise IT Asset Management

Assessment of System Development, Integration, & Sustainment Activities and Certification & Accreditation

Operations Security Management Processes

Operational Enterprise Networks

Trust Management

Enterprise IT Change Management

Identity Management

Centralized Reporting
# Cyber Ecosystem Standardization Efforts

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

- Many standards are XML-based; enabling automation of information exchange
- Several standards support multiple enterprise cybersecurity functions
Kill Chain – Exploit Targets – Courses of Action

Structured Threat Information eXpression (STIX) Architecture v0.3

Why were they doing it?
Why should you care about it?
What exactly were they doing?
What were they looking to exploit?
Who was doing it?
What should you do about it?
Where was it seen?
What could/should have been done to harden the attack surface/vector to prevent the target from being exploitable?
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**
  - Attack Surface Analysis/Threat Modeling
  - Eliminate
  - Mitigate
  - Block from Attack

- **Assess Deployment Risk**
  - Control
  - Alarm for Attack/Exploit

- **Operational Mitigation**
  - Implementation

- **Validate/Verify**
  - Test
Many Capabilities Support the Mission

- Capability
- Hardware
- Capability
- Software
- Capability
- Systems
- Capability
- People
- Capability
- Supply Chain Activities
“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*. 
Attacks & Hazards

Exploitable Weakness #1 (a vulnerability)

Exploitable Weakness #2 (a vulnerability)

Impact from Weakness #1

Impact from Weakness #3

Mission

Impact from Weakness #3

Mission Fulfillment

Mission Fulfillment

Mission Fulfillment
Assurance on the Management of Weaknesses

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

Threat → Threat Vector → Weakness → Mitigate
→ Vulnerability → Eliminate
→ Implementation → Block from Attack
→ Test → Alarm for Attack/Exploit
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.
Structure and Validity Problems
Common Special Element Manipulations
Channel and Path Errors
Handler Errors
User Interface Errors
Pathname Traversal and Equivalence
Errors
Authentication Errors
Resource Management Errors
Insufficient Verification of Data
Code Evaluation and Injection
Randomness and Predictability

cwe.mitre.org
Prioritizing weaknesses to be mitigated

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

Utilizing a Priority List of Weaknesses

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></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SQL Injection</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Insufficient Authorization Controls</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</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>X</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></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Poor Coding Practices</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Technical Impacts – Common Consequences

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 2002, a large number of web servers were compromised using the same SQL injection attack string. This single
Updated Detection Methods Launched 17 Feb 2014

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, 134, 190, 426, 494, 498, 798</td>
<td>98, 120, 131, 190, 426, 494, 805</td>
<td>79, 129, 476, 798</td>
<td>78, 798</td>
<td>78, 798</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modify data</td>
<td>311, 327</td>
<td>78, 89, 129, 131, 129, 190, 311, 78, 89, 129, 131, 190, 311</td>
<td>129, 190, 319, 89, 131, 190, 311, 78</td>
<td>78</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>--------------------</td>
<td>-----------------------------</td>
<td>---------------------------</td>
<td>-----------</td>
<td>---------</td>
<td>----------------</td>
<td>-------------------------</td>
<td>------------------------</td>
<td>-----------</td>
</tr>
<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>78, 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>
</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></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>400</td>
<td>89, 190, 352</td>
<td>665, 798</td>
<td>601, 798, 807</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.
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
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](#)
Software Assurance (SwA) Pocket Guide Series

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)

► Software security testing is not the same as testing the correctness and adequacy of security functions implemented by software, which are most often verified through requirements-based testing that:

  ▪ cannot fully demonstrate that software is free from exploitable weaknesses / vulnerabilities.
  ▪ is not the best approach to determining how software will behave under anomalous and hostile conditions.

Software Security Test Techniques throughout SDLC

Penetration Testing can enhance pre-deployment test outcomes and identify post-release exploit points

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.

Key Practices: leveraging CWE & CAPEC

<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 files or directories</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read application data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read files or directories</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>Modify application data</td>
</tr>
<tr>
<td></td>
<td>CAPEC-63: Simple Script Injection</td>
<td>Modify files or directories</td>
</tr>
<tr>
<td></td>
<td>CAPEC-85: Client Network Footprinting (using AJAX/XSS)</td>
<td>Read application data</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. %3C %3Cscript</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>
When should I focus on Weaknesses and Vulnerabilities?

Focus on Weaknesses:
A type of defect that *may be* exploitable.

Focus on Vulnerabilities:
Something in code that *can actually* be exploited.

Keep Weaknesses from becoming vulnerabilities.
https://buildsecurityin.us-cert.gov/swa

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

https://buildsecurityin.us-cert.gov

Focuses on making security a normal part of software engineering
The Rugged Software Manifesto

I am rugged... and more importantly, my code is rugged.

I recognize that software has become a foundation of our modern world.

I recognize the awesome responsibility that comes with this foundational role.

I recognize that my code will be used in ways I cannot anticipate, in ways it was not designed, and for longer than it was ever intended.

I recognize that my code will be attacked by talented and persistent adversaries who threaten our physical, economic, and national security.

I recognize these things - and I choose to be rugged.

I am rugged because I refuse to be a source of vulnerability or weakness.

I am rugged because I assure my code will support its mission.

I am rugged because my code can face these challenges and persist in spite of them.

I am rugged, not because it is easy, but because it is necessary... and I am up for the challenge.
Software & Supply Chain Assurance: Mitigating Risks Attributable to Exploitable ICT Products and Processes

Joe Jarzombek, PMP, CSSLP
Director for Software & Supply Chain Assurance
Stakeholder Engagement & Cyber Infrastructure Resilience
Cyber Security & Communications

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