Welcome to the

Software Assurance Top 10

If you know nothing about Software Assurance (SwA), but have been told to do it, or you just want to get your software more secure, this is the page for you. Work towards integrating these 10 activities into your program or project lifecycle, and you will be on your way to success.

JFAC regularly updates content in the SwA Top 10 list and supporting pages. Please check back routinely to see the latest information.

Recommendations

Review the below recommendations that will help you start small and continue improving with your SwA efforts:

1. Read the Software Assurance Basics brief.
2. Work towards incorporating the below SwA Top 10 items into your program.

What is SwA?

Software Assurance is a security aspect of Software Engineering which incidentally supports Software Quality. SwA is a continuous activity that aims to ensure software is resilient and secure from attack from cradle to grave.

Why do SwA?

By completing SwA activities, you work towards meeting DoD mandated security requirements. SwA activities also enhance system quality, maintainability, and reliability, as well as lower overall system cost.

Software Assurance (SwA) Top 10

PEO / PM / Project Leads

#1

Contract Language, RFP & Evaluation Criteria

Ensure SwA is accomplished in your program by including language and evaluation criteria in your Proposals and Contracts.

- Example set of documentation (SOW, CDRL, DID, RFP Evaluation Criteria Sections L & M)
- MDA DIDS - Recently released in the ASSIST Database
- Incorporating Software Assurance into Department of Defense Acquisition Contracts

Developers

#6

Foundational Development Practices and Tools

Work to incorporate the below industry best practices and tools when developing software to support secure software engineering.

- Integrated Development Environment – IDE
- Version Control System – VCS
- Software Repository
- Bug Tracking Tools
- DevOps / Build Tools and Continuous Integration Environments
- Configuration Management (CM)

#2

Lifecycle Planning and Budgeting

Forecast the appropriate time and resources to ensure SwA training and execution throughout the lifecycle.

- DoD SwA Across the Lifecycle & Systems Engineering Technical Review (SETR) Events

#7

Secure Design and Development

Develop software that is secure and resilient to attack by including Software Engineering security principles.

- Select foundational training course(s) in
**Cost & Schedule Planning Checklist for SwA Activities**

Learn More →

### #3 Identify SwA Expertise

Identify and build relationships with SMEs to assist in all aspects of SwA.

- SwA SP Services
- SwA Service Providers

Content Coming Soon

### #4 Requirements

Identify and verify secure software development, architecture and operational requirements to thwart attacks.

- Program Protection Plan SwA Countermeasures: SwA-CM-in-PPP.pdf

Content Coming Soon

### #5 Risk Analysis and Decision Support

Identify, understand and prioritize weaknesses that have been discovered during SwA activities for decision support.

- Army Risk Scoring Tool
- Common Weakness Scoring System (CWSS) / Common Vulnerability Scoring System (CVSS)
- Common Weakness Risk Analysis Framework (CWRAF)

Content Coming Soon

### #8 Testing for Software Weaknesses and Potential Vulnerabilities

Use of tools to scan and analyze software to detect weaknesses which may be vulnerable to attack. Different tools and techniques work together to identify different flaws.

- Static code analysis
- Dynamic analysis
- Secure Composition analysis

Content Coming Soon

### #9 Vulnerable SW Components

Review libraries and dependencies for new updates and security vulnerabilities.

- Secure composition analysis — Perform a dependency and CVE review
- Supporting tools
  - Sonatype/Artifactory
  - Maven

Content Coming Soon

### #10 Track and Report Security Bugs

Provide information for decision support. Regularly reporting issues will help decision makers approve, plan or accept the risk for software security issues. This information can come from your normal bug tracking system.

- Content coming soon

Content Coming Soon

**More Info**

Distribution Statement A: Approved for Public Release
Software Assurance Top 10 | Developers

#6 Foundational Development Practices and Tools

Utilize the below Industry best practices and tools to support sound software engineering. As a bonus, these practices and tools support integrating software assurance into the development lifecycle. Since these are core practices, it is likely that projects already have them in place. However, reviewing these practices, and strengthening where appropriate, will make adopting software assurance easier. The result of implementing the following will help identify existing potential vulnerabilities, reduce the number of vulnerabilities inserted during development, empower quick and efficient software changes and subsequent testing to address new threats, and provide early risk decision support all while reducing cost and increasing system defenses. References to products are for example only and do not constitute an official endorsement by the JFAC or an endorsement of any vendor or product over another in any way.

Return to the SwA Top 10

The SwA Top 10 list and supporting pages are under development. Please check back routinely for new content.

Recommendations

Establish or improve upon the below the foundational building blocks and core practices to ensure tools are sufficient to support security throughout software development and testing:

- Integrated Development Environment (IDE)
- Version Control System (VCS)
- Software Repository
- Bug Tracking Tools
- DevOps / Build Tools and Continuous Integration / Continuous Delivery (CI/CD) Environments
- Configuration Management (CM)
- Change Management

Integrated Development Environment (IDE)

An Integrated Development Environment (IDE) provides software developers compilation and code organization tools. Other benefits may include:

- Easy SwA integration through internal or plug-in static analysis tools which can highlight potential vulnerabilities in code. Some plug-ins also provide training and additional information on the issues.
- Integration with bug tracking software and version control systems to help capture code and vulnerability issues in real time and allow teams to prioritize remediation efforts.
- Tool plug-ins which help enforce coding style, naming conventions, and other aspects of code quality.

Example(s)

- Eclipse
- Visual Studio
- IntelliJ

Version Control System (VCS)

Version control systems provide development teams a central location to store code, configuration and other documents necessary for the software development process. Version control is essential to allow multiple developers to code independently, merge code into a single version, support team code reviews, perform bug tracking and reporting, and allow versions to support past, current and test processes. Other benefits may include:

Distribution Statement A: Approved for Public Release
IDE integration so developers can commit (submit) their code into their team repository without using a different application or interface.

Code review tool integration to allow code reviews to draw from specific check-ins in the repository.

**Example(s)**

Open source VCS tools which provide similar services with different source project management structures.

- Git
- SVN
- Mercurial

Hosted repository management systems that provide useful features for managing software source code and resources.

- GitLab
- GitHub
- Bitbucket

**Software Repository**

A software repository is similar in concept, but usually different from a VCS. Software repositories are designed to store software artifacts that are generally much larger than source code files. These artifacts are usually fully compiled, deployable software. Because of the types of items that are stored, artifact repositories generally have different technologies and metadata they rely on and perform storage in a different manner due to the different types of information stored. Other benefits may include:

- Storage of an organization’s documentation, internal releases, or 3rd party libraries required to build the software.
- Management of libraries within a Version Control System. However, it is possible to place the configuration or build files (e.g. package.json for JavaScript applications or Maven POM files for Java) that identify the libraries used under configuration management with the libraries stored within a Software Repository. At a minimum, store both the third-party library and a hash of that library. A software repository is also a good location to store base images for containers.
- Operating System (OS) support for an Application Package Tool (APT, a Package Manager), RPM (a Linux Package Manager), or a Windows App store; or may be focused toward a specific programming language and development libraries (maven-central).

**Example(s)**

Organization/enterprise repository managers

- Sonatype Nexus
- JFrog Artifactory

**Bug Tracking Tools**

Bug, issue, or incident tracking tools allow users and project leads to create, store, and manage issues and work items, such as updating status, criticality and schedule. Many bug-tracking tools are now integrated into more comprehensive project management systems, which provide a holistic view of the software development process underway. Other benefits may include:

- Support for Agile development which heavily utilizes issue tracking systems. Jira is an example of a comprehensive project management system that has integrated issue tracking.
- Hooks into version control software which can monitor development progress.
- Changes needed to support secure software and reduce vulnerable attack-paths are included. They can also provide reports to identify security issues to help decision makers approve needed changes early.

**Example(s)**

Focused bug tracking tools

- Trac
- Bugzilla
- Mantis
integration is a current trend, where the software is built and tested whenever it changes. However, even a nightly or weekly build process that includes automated testing can provide a foundation for adding additional software assurance activities.

Build tools maintain a consistent compilation, assembly and linking process; manage software dependency information; and run unit tests to ensure build and code quality. Other benefits may include:

- Build orchestration tools execute build scripts and run any associated tests. It is external to the compiler or packaging process and helps manage that process while providing a common environment for all developers on the team. An open source example is Jenkins. Note that the build scripts should be placed under Configuration Management, with changes tracked under Change Management. This tool needs to be extensible to integrate capabilities such as Static Analysis Software Testing tools, discussed in another Top 10.
- Build tools can be used in a continuous integration / continuous delivery pipeline environment to integrate and automate activities which support developers and testers while improving software quality and delivery speed.
- SwA tools that perform static, dynamic and secure composition analysis can be included in the automated build environments to identify additional potential vulnerabilities within the static code, the dependent libraries, and the running application.
- A DevSecOps pipeline is an example of a type of environment which will include development and build tools as well as SwA tools resulting in a more hardened, secure and resilient product. For a brief and easy introduction to pipelines, see https://opensource.com/article/19/4/devops-pipeline.

Example(s)

Pipeline tool which supports SwA tools
- Jenkins

Configuration Management

Software Configuration Management (SCM) ensures the software deliverable is repeatable, consistent, predictable, simple and clearly understood. CM ensures knowledge of the exact configuration of the system, including the environment in which it is deployed and the environment in which it is built. Configuration Management is the practice by which known configurations of the system are captured and changes to them are then maintained. It is closely tied to Change Management, which is discussed below. SCM benefits include:

- Since the build and delivery process is orchestrated, the process and the resulting product is clearly understood, such as in a CI/CD pipeline. This supports efficiencies in future product changes. When changes to a version of software are required, the areas of change and impacts of the changes can be more quickly identified and therefore the changes can be more quickly made.
- Additionally, since the product and what it contains is easily understood, it is more difficult for malicious code, dependencies or configurations to be inserted or enacted.

For software assurance, the following are a minimal set of configuration items:

- **Source Code.** All source code that becomes part of the deployed system, including any configuration files and static content such as images and multimedia files. It is recommended that all source code be placed under configuration management, including source for tests and utility tools that the project may develop. Source code is typically configuration managed within a Version Control System (discussed above).
- **Third-Party Libraries.** In modern development, substantial portions of applications are provided by third-party libraries, often open source libraries. Since these are part of the project, they should be placed under configuration management either in a VCS or a Software Repository (discussed above).
- **Build Configuration.** All software, scripts and configuration files that are used to build the product that is deployed should also be placed under configuration management. This will allow the exact same version of the product to be built at any time in the future. If containers are used for deployment, ensure that the base containers used to build the deployed containers are under configuration management.
  - **Example:** JFrog Artifactory as a software repository can hold the build configuration.
- **Deployment Environment Configuration.** It should be possible to recreate the exact deployment environment used by the system through artifacts under configuration management. This could be accomplished by well-maintained and version-controlled documentation. However, a modern practice is to use “infrastructure as code” to capture the configuration of the deployment environment. This practice uses a set of configuration files and scripts that can then be used to recreate the environment.
  - **Examples:** Ansible scripts and Dockerfiles can be used to provision environments and deploy applications within a Jenkins pipeline.

Change Management

Distribution Statement A: Approved for Public Release
Change Management should be tightly coupled with Configuration Management as it is important to know both the configuration of a system as well as why it has changed. Systems should exist to identify and track changes to any item under Configuration Management. Such changes may also require reviews and approval, where software assurance practices such as secure code reviews and penetration testing can be inserted.

**Recommended Tools**

- **Defect Tracking System.** A defect or bug tracking system is used to identify issues within the application that require a change to the source code. Often, defect tracking systems are also used to track enhancements and other non-defect changes. Defect tracking systems often integrate with a VCS such that changes to the source code can be directly linked to the defect or change request that caused them.

- **Issue Tracking System.** For deployed applications, often a Help Desk is utilized to field calls from users when issues occur. Such issues may be defects in the software, which would then be tracked in a defect tracking system, but may also be issues in the deployment environment, such as the failure of a database. A strong issue tracking system will allow for analytics and analysis to be performed to identify trends and root causes of issues.

- **Release Management System.** Releasing a new version of the software represents a significant change that should be managed. Often, formal approval for releases are required, which also represent an insertion point for software assurance approvals. Organizations may integrate such approvals into an IT Service Management (ITSM) process such that releases are coordinated via a Change Calendar.
The best software in the world can be rendered useless or dangerous to friendly forces if attackers can take control of it. Software Assurance is the security component of Software Engineering that helps increase the level of confidence that software functions as intended and is free of known unmitigated vulnerabilities throughout the lifecycle. Our national defense systems will soon be 80% software which is where 80-90% of vulnerabilities have been found over the last decade. Insecure software architecture accounted for 50% of those vulnerabilities. While this section recommends specific classes of tools, it does not recommend or endorse specific products as such choices can be driven by a number of project and organization specific factors.

Return to the SwA Top 10

#7 Secure Design and Development

Recommendations

Use the below resources to find and fix software security weaknesses in your current code and prevent future vulnerabilities.

1. Select foundational training course(s) in secure software development and the hacker mindset
   - Free
     - Hacker 101 & Secure Coding class training & watch the Software Angel of Death lecture video (~1 week to complete on own time)
     - “Secure Software Design and Programming” by David Wheeler (GMU graduate course, SWE-681/ISA-681)
   - Paid
     - Certified Secure Software Lifecycle Professional
     - SEI classes on secure software development – Carnegie Mellon Software Engineering Institute (SEI)
       - Secure Coding in Java
       - Secure Coding in C and C++
       - Examination of Software Architecture: Principles and Practices
       - Secure DevOps Process and Implementation
       - DevSecOps Process and Implementation
       - Security Requirements Engineering Using the SQUARE Method
       - Software Assurance Methods in Support of Cyber Security
     - SANS classes on secure software development
       - DEV522: Defending Web Applications Security Essentials
       - Certified Application Security Engineer (CASE)

2. Choose a Secure Coding Standard

Secure Coding Guidelines are guidelines for writing software code during its implementation that prevent or reduce the risk of failures in software assurance. Examples include:

- MISRA
- Mozilla’s WebAppSec
3. Perform manual secure code reviews

Secure Code Review is a deep-dive and audit of your source code to ensure all security standards and controls are in place. Security is the primary driver for decision making during this review. Please review the below resources as a reference for performing manual secure code reviews.

- OWASP Code Review Guide
- SmartBear Best Practices for Code Review

4. Architect & design secure software

If software and architecture are developed modularly and under solid configuration management, changes to address threats will be quicker and easier. There are architectural analysis tools which help identify overly complicated and interdependent code sections which helps keep code modular to address upcoming changes and understand the impacts of changes which provides clarity and speed. These tools range from the simpler Understand tool which shows your software structure to jArchitect tool which looks for cyclical dependencies to tools which support more features and capabilities such as Silverthread, KDM Analytics Blade RiskManager, and CAST, some of which integrate with other processes and have their own frameworks. A study is being performed in FY20 to provide more information on these tools which will be posted on this portal.

5. Test code for vulnerabilities during development

For additional information about testing code for vulnerabilities during development, see SwA Top 10 item #8 Testing for Software Weaknesses and Potential Vulnerabilities.

6. Check software libraries and components for vulnerabilities

For additional information about checking software libraries and components for vulnerabilities, see SwA Top 10 item #9 Vulnerable SW Components.

More Info

- Common Weakness and Enumeration (CWE). For each software weakness, it lists information on how to implement, architect, test, etc. Provides a wealth of information. The Hacker 101 and Secure Coding class walks through a use example that ties in the CAPEC as well.
- Gary McGraw Book Series on Software Security:
  - Software Security: Building Security In
  - Building Secure Software: How to Avoid Security Problems the Right Way
  - Exploiting Software: How to Break Code (the best defense is to know what the offense is doing)
- Cyber Security Engineering - A Practical Approach for Systems and Software Assurance by Nancy R. Mead and Carol C. Woody
- Rugged Software
- SafeCode
- Software Assurance Pocket Guide Series
  - Secure Coding
  - Architecture and Design Considerations for Secure Software
  - Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses
- Common Vulnerabilities and Exposures (CVE)
- National Vulnerability Database (NVD) - Search for software products with known vulnerabilities (CVEs)
- Common Attack Patterns Enumeration Classes
The SwA Top 10 list and supporting pages are under development. Please check back routinely for new content.

**#8 Testing to Find & Fix Vulnerabilities**

80-90% of breaches are caused by software weaknesses and 50% of those are from insecure software architectures. Combining SwA tools during development with a software security architectural review (which a JFAC SwA service provider can perform) significantly reduces the attack paths and impacts into the software. This will help protect the system from a hacker taking control of a system or just preventing it from executing when it is most needed. Integrating a SwA tool or two will help find software security weaknesses during development - a couple at a time. Liken it to unit testing. The more you find early, the less impact to the software later. Unfortunately, there is no "Silver Bullet" single tool which finds most of the potential vulnerabilities. References to products are for example only and do not constitute an official endorsement by the JFAC or an endorsement of any vendor or product over another in any way.

**Recommendations**

There are a variety of tools and activities that, when used during software development and testing, can provide a higher level of confidence that your software is secure and resilient to attack. This is a journey, so start small and grow over time.

Many of the topics on the Developer side of the SwA Top 10 interrelate. Examples are:

- Plugging in a static analysis tool, mentioned below, into the development environment (relates to both #6 Foundational Development Practices and Tools and #7 Secure Design and Development)
- Reviewing COTS and FOSS components for known vulnerabilities (secure composition analysis mentioned below relates to #9 Vulnerable SW Components)
- Tracking and reporting tool results (relates to #10 Track and Report Security Bugs)

Testing for software weaknesses and potential vulnerabilities is a layered problem that is best addressed through divide and conquer between developers and independent SwA providers. Developers should use static and dynamic analysis tools that review code and runtime execution during development. Issues can then be addressed when the code is fresh in memory and in a small manageable set per week. JFAC SwA service providers can use a larger variety of tools and processes to find issues which directly affect the system’s mission. Using independent SwA provider expertise will reduce the developers’ software security load to a reasonable set of activities which will keep their focus on their main expertise, creating system code. For existing programs just starting in secure development testing, a service provider can also help wade through the large set of findings that a static analysis tool will find the first time it is run on an existing large codebase. This will expedite initial static analysis scans so that the following weekly developer scans will be lightweight and manageable. An independent SwA service provider can also help with a software security architectural review to ensure less rework is required later.

**Tools for Developers**

- **Static Analysis - Static Application Security Testing (SAST)** - This type of tool looks at the written lines of code and their static execution paths to see if there are software weaknesses which could be exploited. This type of tool can be used during development, at your build integration server and in the test environment. Once a developer starts using these tools regularly (daily/weekly), they will find very few bugs. Some issues will be able to be handled quickly and immediately. Others will need to be raised for program awareness and risk decision making about whether to fix, mitigate or disregard the issue and to determine which versions to address fixes. Any issues that are mitigated or disregarded should note the rationale in the bug tracker and/or report.
  - Example tools:
    - Open source tools: SpotBugs with the FindSecurityBugs plug-in, cppCheck, ...
    - Commercial: Parasoft, GrammaTech’s CodeSonar, Fortify, Coverity, ...
  - Note that while these are costly, they come supporting a variety of languages, easily integrate with other development tools, are actively upgraded to handle new security issues, and usually do a more thorough analysis which helps find more vulnerabilities in your code.
Caution, sometimes teams pick tools known to find fewer “false positives”. While this may seem like less issues to deal with, it is probably better to select a tool which finds more possible security bugs and have your developers quickly determine as they go whether they are real or not.

- **Dynamic Analysis** - Dynamic Application Security Testing (DAST) - Runs the executable code and tries to find exploitable points. Since it requires an executable, it cannot be run as early as static analysis.
  - Example tools:
    - Open source tools: Peach (fuzzer), American Fuzzy Lop (fuzzer), Samurai WTF (web), Scuba (database), ...
    - Commercial: AppScan (web), AppDetective (database), ...

- **Secure Composition Analysis (SCA)** (more info in Topic #9) - Checks to see if the library and dependency components supporting the software development effort have known vulnerabilities.
  - Example tools:
    - Open source: DependencyCheck, ...
    - Commercial: ionChannel, BlackDuck, Sonatype, ...

- Website listing of Common Vulnerabilities and Exposures (CVE) vulnerable packages

---

### More Info

- **(U) The State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test and Evaluation 2016 v2**
- The National Security Administration Center for Assured Software Static Analysis Tool Study Reports
  - (U/FOUO) Static Analysis Tool Study Report - C/C++, Phase 1 2012 update
  - (Phase 2 - covering tools not included in the 2012 report)
  - (U/FOUO) Static Analysis Tool Study Report - Java, Phase 1 2012 update
  - (Phase 2 - covering tools not included in the 2012 report)
- NIST Listing of Static Analysis Tools
- Other SwA Testing Types
  - Data Correlation - These tools will combine the results of multiple tests into a single report to support and reduce analysis. Examples are Code Dx and TOIF.
  - Attack surface identification - Software Attack Surface.
  - Software Threat Modeling/Risk Modeling - A simple tool to help an easy method to diagramming the program components and flow. The Microsoft Threat Modeling Tool is an example.
  - Architecture Analysis - Complex architectures hide insecurities and prevent quick changes needed to deal with threats. These tools look for issues such as unnecessary complexity, interdependence and oversized modules. Examples are jArchitect and Silverthread.
  - Coding standards and compliance testing - Following a coding standard and using a tool to check for compliance, such as Rosecheckers, can help increase engineering goodness and reduce insecure practices. Most static analysis tools will also report on coding standard issues.
  - Interactive Analysis - Interactive Application Security Testing (IAST) - These tools are a combination of static and dynamic analysis tools. They run on an executable and mimic a hacker but look for memory and other items as if they were a static tool. An example tool is Contrast Assess.
  - Risk Scoring - Scoring of weaknesses for Risk Decisions - See Topic #4 Risk Analysis and Decision Support. These types of tools help highlight and prioritize risks for decision support.
  - Penetration testing.
  - Manual security testing for specific security and business rules that the other tools won't find (can be separate or included as an expansion of the static and dynamic testing aside from the automated tools).
  - Runtime Analysis - Runtime Application Self-Protection (RASP) - This is a monitoring tool which looks for attacks. An example is Contrast Protect.
  - Binary Analysis - tests for security issues on provided executables where source code is not available.
Testing tools work together like puzzle pieces to build the software security picture. Below is a pictorial view showing how Static Source Code Analysis, Data Correlation, Risk Analysis, Origin Analysis, Binary Analysis and Dynamic Analysis tool fit together to form that picture.
The SwA Top 10 list and supporting pages are under development. Please check back routinely for new content.

**#9 Vulnerable SW Components**

Topic #6, Foundational Development Practices and Tools, discusses how to make sure the software developed for the program is not vulnerable. This topic covers how to test to see if the libraries and software components that software developers re-use (but did not develop themselves) can be checked for known vulnerabilities or out of date versions. Stay up to date by upgrading to newer versions that address vulnerabilities. If you can’t upgrade, find what issues exist and address or mitigate them. References to products are for example only and do not constitute an official endorsement by the JFAC or an endorsement of any vendor or product over another in any way.

Return to the SwA Top 10

**Recommendations**

Perform vulnerable software component identification (secure composition analysis) and software version updates/patches to address known issues.

Check libraries and software components to see if any have known vulnerabilities by:

1. Collecting your list of software libraries and components
   - Note: For example, Dependency Check is an open source tool that has a plug-in for Maven. By putting Dependency Check in the build file, Dependency Check will then output which components have known vulnerabilities. See the bolded information in “More Info” below. This fulfills #1 & #2 in this list.

2. Looking for known vulnerabilities in the list of software
   - Use the MITRE Common Vulnerabilities and Exposures website listing of known vulnerable components or use a tool such as OWASP’s Dependency Check (open source) and Black Duck or Sonatype (commercial) or similar tools (See More Info below).

3. Updating software versions to the most recent possible
   - Software has updates partially because hackers find vulnerabilities within the products and the update is the patch or fix to prevent their activities. Hackers update and share this list of vulnerabilities. If a version cannot be updated, plan to address or mitigate the vulnerability with another method. Keeping software as up to date as possible during the lifecycle will significantly support security.

**More Info**

Secure Composition Analysis (SCA) Tools for Vulnerable Software Component Identification

Three items to check:

1. If any software libraries or commercial off the shelf (COTS) or free and open source software (FOSS) components have known vulnerabilities
2. Additionally, some commercial tools can also test any code that was copied and pasted (code reuse) from external sources to see if they came from a vulnerable library as well as offer replacement components for any vulnerable ones reported.
3. If there are newer versions of software available (which will be more secure due to fixes)

SCA tools check external libraries and components which were downloaded or copied to support program code for known vulnerabilities. An open source example is OWASP’s Dependency Check tool which will take a software bill of materials and list the program’s libraries, dependencies and software components and identifies which ones have known vulnerabilities.

There are separate tools which can help obtain the software bill of materials for SCA tools. Examples below:

- C/C++: Tool example coming soon
- Java: One example is Maven which has plug-in for Dependency Check.
Using Maven separately and in combination with Dependency Check will enable obtaining a list of an application's dependencies, highlighting which ones are vulnerable or out of date. Maven's `versions:display-dependency-updates` command will show if newer versions exist for an application's dependencies.

- Maven with Dependency Check links

- Apache Maven's site info
  - `versions:use-latest-versions` searches the pom (build file) for all versions which have a newer version and replaces them with the latest version

Additional helpful Apache Maven commands (https://www.mojohaus.org/versions-maven-plugin/)

- `versions:use-latest-releases` searches the pom for all non-SNAPSHOT versions which have been a newer release and replaces them with the latest release version
- `versions:update-properties` updates properties defined in a project so that they correspond to the latest available version of specific dependencies. This can be useful if a suite of dependencies must all be locked to one version
- `versions:display-dependency-updates` scans a project's dependencies and produces a report of those dependencies which have newer versions available
- `versions:display-plugin-updates` scans a project's plugins and produces a report of those plugins which have newer versions available
- `versions:update-parent` updates the parent section of a project so that it references the newest available version. For example, if you use a corporate root POM, this goal can be helpful if you need to ensure you are using the latest version of the corporate root POM.
- `versions:lock-snapshots` searches the pom for all -SNAPSHOT versions and replaces them with the current timestamp version of that -SNAPSHOT, e.g. -20090327.172306-4
- `versions:unlock-snapshots` searches the pom for all timestamp locked snapshot versions and replaces them with -SNAPSHOT.
- `versions:resolve-ranges` finds dependencies using version ranges and resolves the range to the specific version being used.
- `versions:use-releases` searches the pom for all -SNAPSHOT versions which have been released and replaces them with the corresponding release version.
- `versions:use-next-releases` searches the pom for all non-SNAPSHOT versions which have been a newer release and replaces them with the next release version.
- `versions:use-next-versions` searches the pom for all versions which have been a newer version and replaces them with the next version.
- `versions:commit` removes the pom.xml.versionsBackup files. Forms one half of the built-in "Poor Man's SCM".
- `versions:revert` restores the pom.xml files from the pom.xml.versionsBackup files. Forms one half of the built-in "Poor Man's SCM".
- `dependency:list-shows` the list of dependencies for the project