Linux Academy

DevSecOps Essentials

Study Guide

John Marx

john.marx@linuxacademy.com

December 28, 2018

Contents

Section 1: Introduction

Overview of DevSecOps

DevSecOps is a portmanteau (or mashup) of Development, Security, and Operations. It is the term used to identify a software development and continuous delivery process used by enterprises to improve time to market and ensure ongoing quality and stability of information and computing systems.

Purpose

The purpose and intent of DevSecOps is to affirm and encourage the mindset that "**everyone is responsible for security**". The Agile "Shift Left" approach improves team autonomy and throughput when developers are empowered. Security decisions at speed and scale require clearly defined and transmitted policies that are well supported through effective automation.

- **DevOps vs DevSecOps**: DevOps is an established process and practice whose aim is to automate the interactions of development and operations to speed software delivery. DevSecOps is an evolving process that calls out the role of Security to place more emphasis on the need to empower developers and cross-functional teams to implement security policy when automating and using DevOps processes.
- **Why DevSecOps Matters**: As technology proliferates, it becomes integral to the ongoing function of society. Software permeates everything we do and the onset of things such as artificial intelligence, the internet of things, and self-driving cars means that software can lead to great benefit or lethal harm. Cyber threats due to malfeasance increase as technology becomes more pervasive. Automated software delivery establishes unattended processes that once were step-by-step with human hand-offs and direct oversight. DevSecOps is the process that defines principles and practices necessary to safeguard software and manage the ongoing threat from hostile forces seeking to exploit vulnerabilities.

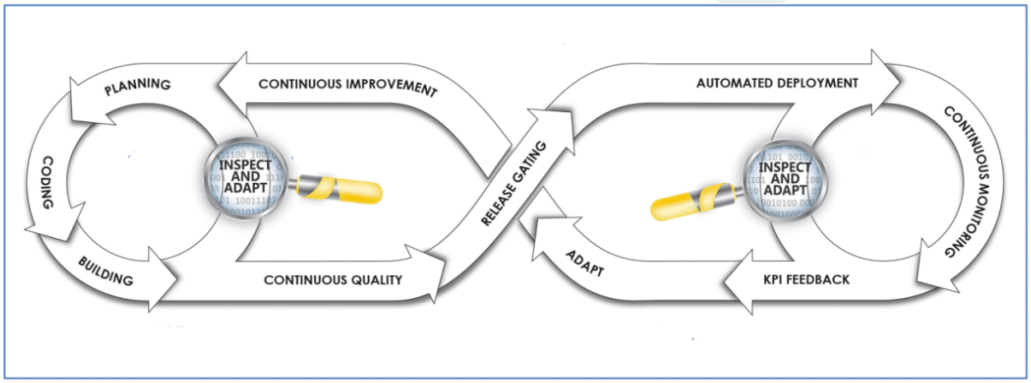


Figure 1.1 : DevSecOps honors the iterative processes of development and operations. By creating an end-to-end iterative process it encompasses Continuous Integration, Continuous Deployment, and institutes a Continuous Deliver discipline.

- **Agile Development**: Agile processes in development conduct iterations through a backlog of features and user stories. Scrum, the most widely used Agile lifecycle method uses Sprints to organize work. Each iteration includes a cross-functional team that designs, builds and tests software.
- **Staging**: Once developers achieve a definition of done, namely code that when tested adheres to specifications, work product is staged for further testing and promotion from lower environments to test infrastructures, staging areas and eventually production environments.

- **Release Management**: Release management is the process through which developed software that has been staged for release is scheduled and released into production in a systematic way to ensure ongoing stability.
- **Monitoring**: When promoted to production, applications and systems are monitored for further vulnerability to ensure that as new threats become known, vulnerabilities in systems previously developed are quickly identified. Dependent upon the severity of the vulnerability, applications may be removed from production or cleansed of vulnerability. Typically remediation involves the development organization as refactoring may be required to update component libraries to versions that have eliminated the threat.

Section 2: Cyber Security Standards and Concepts

- **Cyber Security** is a vast field of study. Qualified security professionals have many years of experience combined with many certifications. DevSecOps is not a complete security curriculum. It is rather a process used to fully integrate security operations into DevOps.
- **Attack Surface** (**Attack Vectors**): The attack surface of a digital system is the collection of points (the "attack vectors") where an unauthorized user (the "attacker") may enter to inject data or to extract data from an environment. Keeping the attack surface as small as possible is a basic security measure.
- **Malware**: Malware refers to malicious software attackers dispatch to infect individual computers or an entire organization's digital network. Malware exploits target system vulnerabilities, such as a bug in legitimate software (e.g., a browser or web application plugin) that can be hijacked.
- **Vulnerability**: A vulnerability is a weakness or deficiency in any system that may be exploited by an attacker to perform unauthorized actions in a computer system.
- **Common Vulnerabilities and Exposures** (CVE): A dictionary-type list of standardized names for vulnerabilities and other information related to security exposures. CVE aims to standardize the names for all publicly known vulnerabilities and security exposures.
- **Common Vulnerability Scoring System** (CVSS): A free and open industry standard for assessing the severity of computer system security vulnerabilities. CVSS attempts to assign severity scores to vulnerabilities, allowing responders to prioritize responses and resources according to level of threat.
- **Security Content Automation Protocol** (SCAP): A U.S. standard maintained by the National Institute of Standards and Technology (NIST). The OpenSCAP project is a collection of open source tools for implementing and enforcing this standard.
- **Center for Internet Security** (CIS): The CIS provides security benchmarks and the National Checklist Program (NCP), defined by NIST. They offer guidance on the security configurations of the operating system, database, virtualization, framework, and applications. In addition to CIS Benchmark documents, CIS also provides tools to infrastructure or operation teams for secure configuration scanning. The CIS Security website provides related security configuration scanning tools to download.

- **Malware and Vulnerability Scanners**: Scanners may be deployed to hosts on a network and run in a memory resident mode to monitor activities happening in real-time. By monitoring multiple sensor points, vulnerabilities may be logged so that DevSecOps stakeholders become aware of the need for remedial action. Scanners are recommended to interrogate new and existing software to determine whether any system or application component may be infected by threat actors or attackers. Scanning may be performed as dynamic scanning or static scanning.

- **Dynamic vs. Static Scanning**: Dynamic scanning is a method of code analysis that identifies vulnerabilities in a runtime environment. Automated scanning tools may provide for analysis of applications in which you do not have access to the original source code. A dynamic test will monitor system memory, functional behavior, response time, and overall performance of the system. Static scanning is a method of analysis performed in a non-runtime environment. Typically, a static analysis tool will inspect program code for all possible run-time behaviors and seek out flaws and potentially vulnerable code. Having originated and developed separately, static and dynamic analysis are not in opposition to one another. There are strengths and weaknesses associated with both approaches and many DevSecOps processes benefit from the use of both methods.

- **Cloud Security Alliance** (**CSA**) **Cloud Controls Matrix** (**CCM**): The CSA (Cloud Security Alliance) Cloud Controls Matrix (CCM) consolidated most security compliance methods into one matrix called CCM. CCM includes all security compliance controls such as ISO, FedRAMP, and NIST 800-53. It defines the control ID used to uniquely identify vulnerabilities. The key benefit of referring to CCM is that we can focus on one aggregate source of security compliance regulations. CSA also provides a Consensus Assessments Initiative Questionnaire (CAIQ). This questionnaire for cloud consumers or cloud providers is a means of security self-assessment.

- **Open Web Application Security Project** (OWASP): The Open Web Application Security Project is an online community project that provides freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security. The OWASP Foundation came online on December 1, 2001. It was established as a not-for-profit charitable organization in the United States on April 21, 2004. OWASP is an international organization and the OWASP Foundation supports OWASP efforts around the world.

- **Federal Information Processing Standards** (FIPS): The FIPS defines minimum security requirements for the use of cryptographic modules. DevSecOps professionals recommend that the Security Requirements for Cryptographic Modules be referenced to understand what cryptographic modules may be considered safe, legacy, or weak. Developers wishing to learn about cryptographic modules may consult the following resources:

    - OWASP Cryptographic Storage Cheat Sheet
    - OWASP Guide to Cryptography
    - OWASP Key Management Cheat Sheet

- **National Checklist Program** (**NCP**) **Repository and National Vulnerability Database** (**NVD**): The NCP repository provides secure configuration for specific software components. DevSecOps stakeholders may use the NCP to search CIS (Center for Internet Security) information on particular software products. NCP provides metadata and links to checklists of various formats, including checklists that conform to the SCAP (Security Content Automation Protocol). The NIST provides an online search tool as well as data feeds that are used

by many SCAP validated tools to provide the intelligence required for advanced DevSecOps monitoring. NVD is an acronym used by security professionals to refer to this NIST repository and other data stores provided by the NIST to security professionals and vendors.

Identity and Access Management

- **DevSecOps** pipelines require that many of the processes required for software development, testing, and deployment be automated. Each server and tooling application requires that security practices be implemented to reduce attack vectors and prevent malfeasance. Specific practices are unique to each system but can be categorized as server hardening, application hardening, and ongoing identity and access governance.
- **Identity and Access Management** (IAM) is the security and business discipline that governs individuals and systems access to computing resources and assets. Most modern software applications and frameworks have implemented specific ways authentication by individuals is governed. Systems that utilize application-to-application integration through API's (Application Program Interfaces) also have frameworks through which they govern access. The process of hardening servers and applications in a DevSecOps pipeline requires specific practices based on the particular tooling being used.
- **Server Hardening** is the security practice of enhancing a server platform's security through a variety of means. Benchmarks from CIS, and applications such as OpenSCAP may be used to provide review of server vulnerabilities and identify appropriate means for risk mitigation. A server must be hardened before the applications and tooling run on the server can be secured.
- **Application Hardening** is the security practice of enhancing an application or framework's security through the recommendations of the application or tooling provider. Most DevSecOps tooling involves automated integration with other third-party systems through Application Program Interfaces (API's). Hardening an application includes the initial implementation of security assets but also the ongoing governance of those assets over time.
- **Identity Repositories** are systems such as Active Directory or LDAP Servers that are used to maintain one source of truth for all users and groups within an enterprise.
- **Access Keys** such as RSA keys are often employed for server access such as when Ansible needs to escalate privilege to configure remote servers.
- **Signatures and Certificates** are used to allow programs to authenticate the source and authenticity of digital assets.
- **Vaults** are employed to manage secrets and ensure that access credentials are stored in encrypted forms thus preventing access by attackers.

Secure Software Onboarding

- **Software Onboarding** is the process a developer uses to acquire the third-party components needed to develop and compile an application. With the onset of open source, many third-party component libraries and frameworks are acquired from public repositories.Off-Premise repositories are not administrated and maintained by the developer's organization.
- Attack Vectors are often exploited in public systems. Since the systems are not on-premises, they must be hardened and maintained by the controlling organizations that provide them.Highly regulated industries require levels of security and protection that may exceed those practiced by the organizations maintaining public repositories.

- Malware exists in most public repositories and some of the most popular components in use today are known to be vulnerable. If the software in public repositories is transferred to internal systems, the malware is brought into an on-premises infrastructure. Once inside the private infrastructure, malware can be unknowingly propagated to production where vulnerabilities are further exploited by attackers.

- **Firewall** systems prevent access to internal systems from outside attackers. At the time of onboarding, software should be scanned for known vulnerabilities. On-premises systems must be hardened to prevent unwanted malfeasance. On-premises repositories must be scanned on an ongoing basis and monitored to ensure threats do not penetrate the perimeter security in place.

- Large organizations have teams of developers using shared on-premises repositories. The practice of **Source Code Control and Source Code Management** is needed to allow more than one developer to work on the same code-base.

- **Check-in and check-out** processes track code versions and allow the practice of branching from trunk, facilitating sub-releases that are identified for release management.

- **Continuous Integration** is the practice that encourages developers to check-in their code daily so that a shared build may be done to ensure one change does not introduce a defect into the codebase.

- Once tested, code that is staged may be further evaluated for release by automated or manual systems. Once approved for release **configuration management** tooling may deploy the software into production environments and ensure that systems-level dependencies are accommodated to assure proper function of the application. Oftentimes these automated processes allow no opportunity for human review or audit of the code that makes up the completed application.

- Scanning is used to interrogate the software and determine whether known vulnerabilities and malware are present. Since much of the work product is from third-parties, processes to source software from **trusted sources** should be employed.

- **Clean repositories** are those stores of source and object code that have been subject to a rigorous process of scanning and remediation. They are also monitored on an ongoing basis to ensure new vulnerabilities are not introduced nor go undetected. The infrastructure supporting the clean repository must be hardened to prevent intrusion.

- External or off-premise repositories are often certified by various reputable organizations signaling to their consumers that they have been managed in accordance with standard security practices. Internal or on-premises repositories are those that are maintained by specific enterprises and ensure security compliance through their own processes and practices. An example of an external repository might be GitHub. An internal repository would be something like GitHub Enterprise which is an on-premises implementation of the popular external repository.

Clean Repositories

- **GitHub** is a useful and widely used off-premises repository used by many organizations. We will use GitHub as an example of a public repository because it is one of the most well-developed repositories in use today.

- **Source Code**: The files stored in GitHub are often application source code and thus valuable intellectual property. Copying the code may enable other companies or even nation-states to quickly develop derivative applications, saving years or even decades of development time and leveraging trade secrets without paying licensing fees. Hackers might also steal code to resell it on the Dark Web.

- **Attack Vectors**: Hackers can study application source code to gain insights into how to attack the software when run in production. Stealing source code gives them time to look for vulnerabilities that would be more difficult to detect through penetrations of production systems. Attackers can even run code in production on their own systems and test attacks against it, refining attacks for speed, stealth, and effectiveness. State sponsored cyber terrorism and global corporate espionage are well-funded efforts that retain professionals to develop means of intrusion and attack.

- **Login Credentials**: Sometimes files checked into GitHub may inadvertently contain login credentials for other internal or external services. Microservice architectures as well as the increased use of software-as-a-service (SaaS) means that many application components have to authenticate and establish a secure session with an application program interface (API) on another system.

- **Secrets**: Secrets are data elements such as login username and password values. While it is a bad practice, some developers have cut corners and embedded login and authentication data in the source code or supporting files checked-in to GitHub. When hackers gain access to the code, they can gain access to related services, giving them the opportunity to steal more data and disrupt operations.

- **Unauthorized Access**: In many cases, developers are granted access to company repositories from their personal or corporate email accounts. These email accounts may be left vulnerable, and some organizations do not purge old email accounts in a timely manner when developers terminate their employment. Another weak practice is when developers are granted access to all of the company's repositories instead of just what they need for their projects. These deficient policies and procedures, especially over a long period of time, create a wide-open attack surface for attackers to exploit.

- **Insider Threats**: A lack of proactive monitoring can allow malicious insiders to easily hide abnormal activities. A single developer accessing many repositories could be an early indicator of insider threat, and such behavior should be detected and flagged. Unfortunately, due to limited resources, many organizations do not monitor the usage of GitHub and other external repositories. This leaves bad actors opportunity over an extended period of time to continue their intrusion and theft.

- DevSecOps practitioners can take practical steps to tighten the security involved with their use of public repositories.

    - **Clean up login credentials**: Developers should be careful with their GitHub login credentials. It is best to limit access only to those developers who need to be involved in a project. When developers leave a project, their credentials should be revoked.
    - **Maintain and administrate repository settings**: The software behind GitHub, the software version control program, was originally developed for managing development of the Linux kernel. Both Git and GitHub are widely used in open source projects. Some developers, particularly those that contribute to open source projects, treat all GitHub

repositories as public, whether they're open source projects or not. DevSecOps practice requires that your organization's GitHub configuration be maintained and administrated on an ongoing basis to ensure that access isn't any broader than needed.

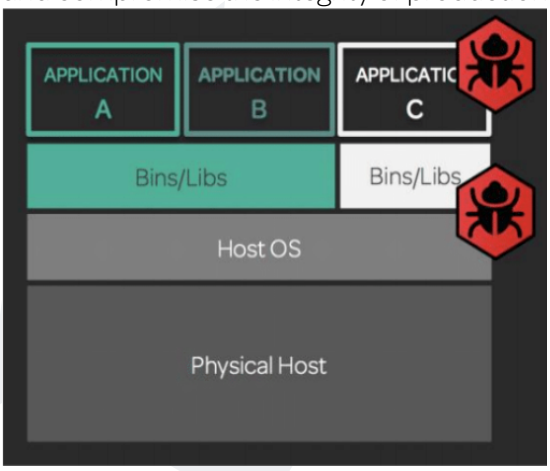
- **Monitor GitHub for suspicious activity**: Automated and manual monitoring of GitHub can detect a sudden spike in source code check-ins. This monitoring should also detect the check-out of an unusually large amount of source code. It is also feasible to monitor and detect logins from unusual locations or logins or requests from users outside the organization.
- **Aggregate and analyze GitHub logs**: GitHub logs can be aggregated and uploaded on a regular basis for further analysis. Tools are available for parsing logs and analyzing activity against a baseline of what would be considered normal use. These practices can be automated so that constant vigilance is performed. When unusual activity is detected, it can be reported through dashboards or even messaged to stakeholders as a means of providing early warning of potential threats.

Secure Containerization

- **Containers** are specially formatted files that contain executable application programs, modules, and the code libraries that they depend upon.
- **Isolation**: Containers isolate an application from other applications, from the operating system, and hardware. Containers help ensure that the versions of libraries within the container do not mandate patching and upgrades that might affect other applications in other containers. In virtual environments, the libraries and dependencies of an application were oftentimes shared with all of the other applications on a particular virtual server.
- **Process Restrictions** use root/non-root dichotomy. This makes it difficult to provoke system level damages during an intrusion, even if the intruder manages to escalate to root within a container because the container capabilities are fundamentally restricted.
- **Device and File Restrictions** further reduces the attack surface by restricting access by containerized applications to the physical devices on a host, through the use of the device resource control groups (cgroups) mechanism.
- **Ephemeral Nature of Container Images**: Container images are ephemeral. Each container has its own file system and can not write to the host file system unless writes are committed. Committing changes tracks and audits revisions made to the base images as a new layer which can then be pushed as a new image for storage in Docker Hub and run in a container.
- **Audit Trail**: The audit trail is important in providing information to maintain compliance. It also allows for a fast and easy rollback to previous versions if a container has been compromised or a vulnerability introduced.
- **Docker Hub** is the world's largest public repository of container images with an array of content sources, including container community developers, open source projects, and independent software vendors (ISV) building and distributing their code in containers.
    - Docker Hub allows you to:
        - Search and browse for millions of container images
        - View image popularity, vendor source, and certification to inform your selection
        - Access free public repositories to store your images and share with the community

- Choose a subscription plan for private repositories to limit access to your images
- Use Autobuilds and Webhooks for easy integration into your DevOps pipeline

- Docker Hub is a public repository. It contains public, private, and official images.
  - **Public images** are shared amongst a broad community.
  - **Private images** might be limited to a single organization or even project within an organization.
  - **Official repositories** are **certified** repositories from independent software vendors (ISV's) and Docker contributors such as Canonical, Oracle, RedHat, and others.
  - Just as a repository can contain malicious source code or infected libraries, Docker containers may likewise be tainted. When they are replicated broadly throughout on-premises and off-premises infrastructures, they can carry the malware with them and compromise the integrity of production systems



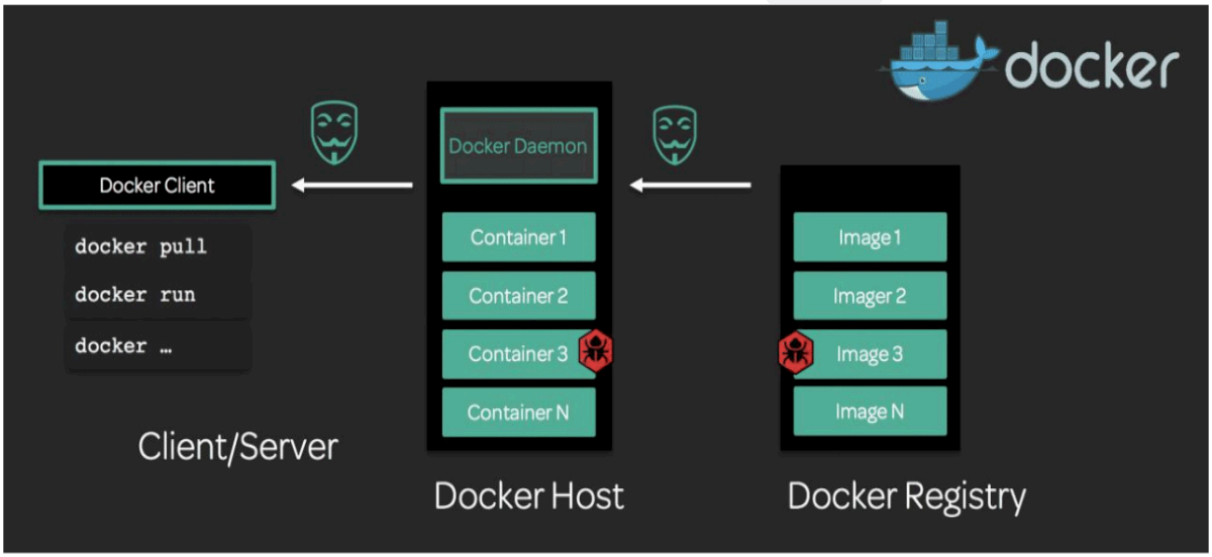


Figure 3: The Docker architecture includes a registry, the Docker Daemon and a Docker Client application. Malware may exist in registries and be propagated by the Docker Daemon if left undetected.

Docker Trusted Registries

- **Trusted registries** are secure managed repositories. A **registry** is a data store that contains **metadata** (i.e., data about data). A repository contains the actual content of interest. Trusted

registries use a **digital signature** to indicate which content in the repository is safe. Software publishers digitally sign their content when it is published. Developers can use systems that verify digital signatures prior to downloading content.

- **Image signing** and **vulnerability scanning** allow operations teams to protect the contents of containers.Metadata may contain information about the author, the **bill of materials**, and whether or not there is a **critical vulnerability**, which can help ensure that a container is safe at the time of onboarding. Automated insights enable organizations to meet compliance requirements and prevent security breaches.
- **Policy-based image promotion** accelerates the DevSecOps pipeline by providing the data needed for automated or manual promotion. The gating process can be configured to require images to pass security scans. **Policy-driven automation** improves scale and speed by accommodating hundreds or thousands of container images.

Docker Notary

- **Docker Notary clients** pull metadata from one or more remote Notary services. Some Notary clients also push metadata to one or more Notary services. A Notary service consists of a Notary server, which stores and updates the signed **TUF metadata files** for multiple trusted collections in an associated database.
- A Notary signer stores **private keys** and signs metadata for the Notary server. Docker Notary controls server access through strict authentication and open SSL security. Clients upload metadata for their container image. Once the data is verified, the server sends approval for the container image to be signed. **Encrypted keys** are used for the signature. Notary Server is the **source of truth** for the state of a **trusted collection**. Timestamps and checksums are used as part of the signature to prevent forgery.
- **Docker Bench** is a software tool that analyzes server infrastructures and Docker components. Docker Bench uses the **CIS Docker Benchmark** to make recommendations on server hardening and proper Docker configuration. Since new vulnerabilities are created and discovered over time, it is important to consult the benchmarks as they are revised and updated. Most importantly, tools that scan and detect based on configuration files that use benchmarks must be updated to the most recent release in order to be effective.
- In large enterprises, the engineering and operations departments create container images that include the necessary tooling for ongoing detection and remediation. Since new vulnerabilities often come from unprotected sources, teams must maintain certified repositories.
- **Trusted registries** are the optimal source, and teams must carry out careful onboarding when using unproven sources. Docker Bench can be run regularly to detect configuration flaws.
- Some of the recommendations found in the Docker Benchmark are:
    - Creating users within the Docker group is a known means for unwanted process escalation. This allows privilege escalation on a spawned container process, which can enable a user to gain root access.
    - Administrators should audit the file system against the Docker directories to receive early warning about unusual access or updates to system logs.
    - Limit open ports as attack vectors through network bridging. Containers allow network bridging to be limited to only those ports necessary for the particular application within the container instance.

    - Since SSL authentication has been deprecated, Docker Bench recommends that the Docker daemon be configured to use **TLS** (**Transport Layer Security**) authentication.
    - When running a container, you can set a ulimit to ensure that the container is limited to a specified maximum number of open file descriptors.
    - Containers should be prohibited from acquiring new privileges.
    - When configuring the Docker daemon, Linux file security must be strictly enforced.
    - The Docker daemon runs as root, and while Linux allows processes to be configured such that their spawned child processes also run as root, this is a vulnerability.
    - Manage file permissions. The 644 octet grants read/write privileges to the owner (in this case, root) but only read privileges to group-and user-level processes.
    - Network sockets carry a similar security setting as file descriptors when permissions are set.

- **Docker Swarm** is a native clustering solution from Docker that can be used for **container orchestration**.
- The Docker Bench includes a review of Docker Swarm hardening recommendations.
- Docker Bench recommendations for maintaining **secrets** in a Swarm cluster include the use of vaults and encryption.
- The Docker content trust system uses **The Update Framework** (**TUF**) to enforce content trust.
- Container image signing and verification policies can be set to allow the use of only trusted repositories.
- The following commands can be configured to require content trust:
    - `docker pull`
    - `docker push`
    - `docker build`
    - `docker create`
    - `docker run`

Section 3: Securing The Automated Build

Automated Provisioning

- **Provisioning** means providing something for use. In DevSecOps, the process of providing the developer with the software needed to develop an application is automated. It's important to ensure that the same software the developers use is provisioned in the test, staging, and production environments. Complex systems require automation to ensure that provisioned processes and components remain uniform and managed.

- A goal of Agile development is to give developers greater autonomy. **Shift left** is a process change in which operations and engineering teams shift processes upstream to developers. Anything "as a service" allows consumers to request and receive deliverables on demand via a self-service portal.
- **Platform as a Service** (**PaaS**) offerings provide developer platforms and infrastructures via self-service tools and automation. Enterprise architects evaluate and test software components for production use. PaaS systems facilitate the development of templates (predefined configurations that comply with policy). When developers require specific frameworks and infrastructures, they can select from available templates.
- **Infrastructure as Code** simplifies the configuration and deployment process and reduces maintenance.
- **Source to Image** (**S2I**) is a framework for building reproducible container images from application source code and dependent component libraries.
- **OpenShift** is an open-source PaaS offering from Red Hat. As containers and clustering have grown in popularity, OpenShift has adopted Docker and Kubernetes as standards. Now that OpenShift can be used to deploy and manage production environments, it is considered a **container orchestration** tool instead of a PaaS. OpenShift is a fully automated container orchestration system that meets developer provisioning needs and automates S2I as workloads are promoted from development to test, staging, and production.

Securing The Automated Build

- **Jenkins** is a popular open-source build automation system. Jenkins originated from the open-source Hudson project. Jenkins is highly extensible and supports hundreds of plugins that handle build tasks. Jenkins supports Continuous Integration by integrating with source code management tools. Jenkins supports **webhooks** that automatically trigger a build process when developers check in modified application source code.
- A **commit** starts the process when the developer checks in code. The Continuous Integration (CI) server launches a build process. Automated unit tests are typically part of the build process. Developers may review the build status via their IDE or the Jenkins console. The build server delivers build artifacts to the appropriate target, typically a staging server.
- **Java** builds typically involve a repo that contains needed component libraries. Apache **Maven** is an open-source tool often used by Java teams. When Maven performs a build, it automatically downloads libraries from **Maven Central**. Maven Central is a public repository maintained by Sonatype.
- Most enterprises configure a repository inside their firewall using products like **Nexus**. Attack vectors exist at the point of code check-in and anywhere an API session is established to pull code. Malware and vulnerabilities exist in most component libraries pulled from third-party repos. Since it is necessary to use vulnerable code, vulnerability detection and monitoring must be done after the build process.

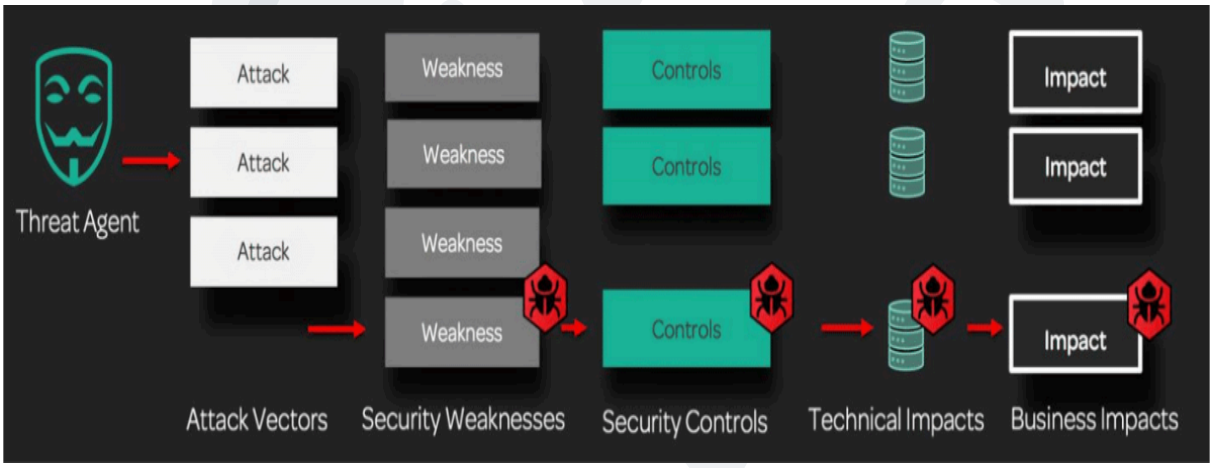
Vulnerability Detection and Remediation

- The **OWASP Dependency Checker** is one open-source tool that detects and reveals vulnerabilities and their severity levels.

- **OWASP Dependency Check** is an open-source scanning system. Dependency Check can be run from the command line or through an automated build program such as Jenkins. OWASP Dependency Check maintains a downloaded CVE database from the NIST NVD. OWASP Dependency Check provides human-readable and machine-readable reports. To enforce security policy, automated builds can be configured to fail when policy thresholds are exceeded.
- **Prioritization** is vital to an effective security strategy. It is impossible to eliminate vulnerabilities. The severity of a vulnerability indicates its **potential threat**. The **actual threat** factors in its context. The key factor for defining and automating a security policy is business impact. Automation tolerates vulnerabilities but prevents critical vulnerabilities from promoting to upper environments until they have been **remediated**.

OWASP Risk Management

> "Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention."



> "Sometimes these paths are trivial to find and exploit, and sometimes they are extremely difficult. Similarly, the harm that is caused may be of no consequence, or it may put you out of business. To determine the risk to your organization, you can evaluate the likelihood associated with each threat agent, attack vector, and security weakness and combine it with an estimate of the technical and business impact to your organization. Together, these factors determine your overall risk."

- OWASP Top-Ten Risks
    1. Injection
    2. Broken Authentications
    3. Sensitive Data Exposure
    4. XML External Entities (XXE)
    5. Broken Access Control
    6. Security Misconfiguration

7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring

- DevSecOps Recommendations to Mitigate Risks

1. Automate scanning throughout the pipeline.
2. Create well-defined security policy.
3. Implement rigorous gating.
4. Reduce the attack surface of DevOps infrastructures.
5. Continuously monitor deployed applications.
6. Establish traceability back to development.
7. Practice ongoing hygiene.
8. Utilize automated Configuration Management.
9. Establish automated roll-back and recovery.
10. Practice Continuous Improvement with ongoing inspection and adaptation to accommodate security enhancements.

Secure Staging

- **Sonatype Nexus** is an open-source repository management system. Build artifacts, produced by programs like Maven, are scanned as part of the automated build pipeline. Once a build output is produced, it is important to push the compiled file to a secure repo. A private on-premises or hardened cloud-based repo server is needed to prevent unauthorized access to or tampering with executable binaries. The use of a repo for build artifacts ensures that test, QA, and staging environments all pull the same file, maintaining uniformity and control of the deployment pipeline.

- The **Nexus** repo manager has a secure API that allows the Jenkins pipeline to push artifacts to the repo. Once in the repo, artifacts are protected, and the attack surface is limited to authorized and authenticated endpoints. QA (Quality Assurance) and UAT (User Acceptance Test) platforms can then pull the executables when needed. Container images can be stored in Nexus to ensure that components and binaries are packaged with build executables. Once a build artifact is stored, all upstream environments use the original binary, even when it is promoted to production.

- **PaaS** and **ARA** tooling depends on a stable repo for fast-and-frequent releases. Automated release and promotion to test environments prevents human error and other factors from disrupting uniform and consistent platforms.

Section 4: Release Gating

- **Capital One** is a good example of a large organization that practices DevSecOps. Capital One Senior Engineering Fellow Tapabrata Pal published an article on Medium.com that forms the basis for this lesson. The article is titled *Focusing on the DevOps Pipeline* and lays out the design principles (the 16 gates) that Capital One uses to develop DevSecOps pipelines.

- **Release gating** is a means of control that involves checks and balances in large enterprises. Authorized stakeholders responsible for governance, risk management, audit, and compliance review build artifacts prior to approving a software release for deployment. While release gating may be done at various stages of the pipeline, it is most often performed in pre-production.
- The 16 Gates are a set of software quality attributes that should be evaluated as part of a governed deployment process. The gates in bold are those considered to be particularly important to security.

    1. **Source code version control**
    2. Optimum branching strategy
    3. **Static analysis**
    4. At least 80% code coverage
    5. **Vulnerability scan**
    6. **Open source scan**
    7. **Artifact version control**
    8. **Auto provisioning**
    9. **Immutable servers**
    10. Integration testing
    11. Performance testing
    12. **Build deploy testing automated for every commit**
    13. **Automated rollback**
    14. Automated change order
    15. Zero-downtime release
    16. **Feature toggle**

Section 5: Continuous Delivery Release Automation

Automated Deployment

- **Continuous Delivery Release Automation** (**CDRA**) refers to the process of promoting application workloads to upper environments with minimal or no manual intervention. Automated deployment systems are typically employed in all post-build environments, including test, QA, UAT, pre-production, and production. CDRA systems are categorized into two domains: **Application Release Automation** (**ARA**) and **Application Release Orchestration** (**ARO**).
- **Automated Deployment** installs build artifacts on upper-level environments, including production systems. Automation is key for improving throughput, controlling outcomes, and ensuring consistency. Once systems have cleared gating, tooling triggers call programs that prepare the infrastructure and deploy or install the application so that it can be executed immediately.

- **Automated Deployment tools** vary widely in capability and implementation. DeployHub is an open-source solution that is available as a hosted SaaS platform or an on-premises implementation. DeployHub integrates with Jenkins build automation, the Nexus Repo, and Ansible playbooks for deployment.

Configuration Management

- **Configuration Management** is the practice and tooling required to fully automate the process of configuring target environments and deploying application workloads to them. While Configuration Management is useful at all stages of provisioning, it is especially important when dealing with production platforms. Configuration Management is an automated approach that replaces traditional scripting and improves control and consistency when deploying applications. Configuration Management, within this lesson, is focused on the domain of Automated Deployment.

- **Ansible** is an open source configuration management tool from Red Hat. Ansible utilizes playbooks to deploy both systems-level and application-level components to target environments. Ansible uses YAML files known as playbooks to automate deployments. Playbooks have the ability to test state on target platforms and only perform those configuration and installation steps necessary to enable the workload. Ansible performs its work in a predefined chronology to ensure dependencies are met. **Infrastructures as Code** (**IaC**) is the key aim of tooling such as Ansible.

Production Monitoring

- **Continuous monitoring** is the ongoing, automated evaluation of workloads to assess their health and function.

- **Production monitoring** may include threat detection, vulnerability detection, and malware prevention. Since new threats and vulnerabilities are developed and discovered over time, it is important to monitor applications throughout their entire lifespan, not just while they're under development. When applications need to be refactored or upgraded with patched third-party libraries, feedback loops and automated reporting notify the development team that vulnerabilities or other defects require attention.

- **Static scanning** is used to test for new vulnerabilities that have become known since an application's release into production. Scanning tools use container image manifests and build time dependency lists to determine which known vulnerabilities are present in compiled applications and container images. Due to ongoing changes to operating systems, binary libraries, and third-party software components, applications need to be continually upgraded. Scanning is fundamental to hygiene, and as applications are redeployed, automated pipelines ensure persistent security.

- **Dynamic scanning** uses black box security testing methodology. A black box test involves testing an application from the outside in while it is running in production. In Dynamic Application Security Testing (DAST), a security analyst attempts to hack an application in the same way a cyber attacker would. Unlike Static Application Security Testing (SAST), DAST does not involve examining source code to identify vulnerabilities from the inside out.

- **Penetration tests** (**pentests**) are a form of DAST that use external programs to interrogate applications through their exposed API and HTTP endpoints. Penetration tests simulate automated cyber attacks on production infrastructure. Penetration tests detect common vulnerabilities such as injection, cross-site scripting, and flaws in authentication and identity and access management (IAM).

Dashboards and Automated Vulnerability Detection

- **Dashboards** provide continual updates on the health of applications running in production. Dashboards aggregate and analyze logs, providing a stream of real-time data that is presented visually based on predefined configurations. Extensible dashboard software systems support plugins that can be configured according to user preferences. Dashboards also let you set up real-time alerts for system events that require immediate attention.

- **DAST** and **SAST** tools can provide logs for aggregation and monitoring in dashboard systems. Many dashboard visualization systems generate benchmark thresholds of normal performance and report anomalies that exceed normal limits.

- **Machine Learning and Security Analytics**: Advanced machine learning and artificial intelligence algorithms can be used to score workloads over time and provide alerts when threats increase.

- **Traceability** refers to the process of communicating information about threat detection back to stakeholders, including the development team. When release management and source code control is established to track major releases and sub-releases of applications, vulnerability reporting can be traced back to the specific source code being evaluated. Alerts and threat detection can be used to automatically roll back vulnerable systems when new releases appear flawed. Configuration management provides an automatic way to upgrade and downgrade applications when necessary.

- Development teams use **Agile backlogs** to record user stories. Production monitoring systems often provide a way to create tickets and integrate with Agile lifecycle management tools. Up-to-date feedback from production systems helps with continuous adaption and improvement.