Woldia University

Institute of Technology

School of Computing

Department of Software Engineering

Software Engineering Tools and Practices

Individual Assignment

Name: Aklilu Mengesha

Id no: 1300244

Submitted to Esmael M.

Submitted Date: March 20, 2024

DevSecOps

1.What are Software engineering problems which was cause for initiation of DevSecOps.

- **Lack of Security Skills**

  Developers, architects, scrum masters, and other key players in an organization should have the right vocabularies and skills. By vocabularies, we mean some common knowledge or skillset, or a common understanding, such as a knowledge of how to write secure code. In our experience, this lack of a common vocabulary often manifests in three ways: The business lacks security skills, developers lack security skills, and/or auditors lack security skills.

- **Lack of Quality**

  Security is integral to quality. In our observation, lack of quality is often associated with the security team getting involved too late, a lack of confidence in the release, and system complexity.

- **Organizational Barriers**

  If you're not sharing your DevSecOps journey across the organization, from concept to product, you should expect problems since you won't have a clear understanding of the business needs your software needs to address. You might not even have a clear vision of the customer's needs and the environment in which the customer operates. Communication is key to breaking down organizational barriers, but often different units inside an organization use different communications tools, structures, and goals.

  To begin to break down these barriers, briefly document your journey from idea to product. Look at points of interaction among the various components of your organization. Educate executives who likely don't know the details of the DevSecOps process. Build connections and a culture that reinforce the sharing of the same goals and vision. Often, poor stakeholder collaboration, difficulty integrating pipeline security, and an unwillingness to make security a priority stand in the way of successful DevSecOps implementation.

- **Lack of Security Assurance**

  How do we know that the security practices we've adopted for our development lifecycle and built into our software are adequate and appropriate for the business purpose we're trying to address? Addressing this challenge can be hard, especially when your industry, business, and/or project lacks security assurance.

2. What is DevSecOps?

DevSecOps stands for Development, Security, and Operations. It is a software development approach that emphasises on integration of security and operations in the software development process. It involves the collaboration of the developing team, testing team, security professionals, and operations team.

3. Briefly explain DevSecOps lifecycle?

DevSecOps is an agile culture of technical best practices which helps to reduce failures between the development, security and infrastructure team. It requires a continuous flow of ordinary execution and appropriate tools.

These are vital stages in devsecops.

1.Planning stage: Identify the threat model and its risks, security requirements and tool deployment strategies.

a. Security requirements: technical specifications needed to implement a project/application:

i. Authentication and authorization: technical definition of how the user will enter the application and infrastructure with access criteria (granulated permission);

ii. Privacy and data protection: specification of encryption (in transit and at rest of the data) and data retention policy — following the compliances agreed (regulated) with the stakeholder;

iii. Integrity: ensuring that data is not tampered;

iv. Availability: ensuring that the application is always available (online) if the workload is overwhelmed by requests, network bottlenecks or electronic failures. Redundancy, failover and disaster recovery are considered availability strategies;

v. Auditing and monitoring (logging): systematic and rigid analysis to check that compliance and procedures are being carried out and followed. Log retention policy for possible later analysis, keeping a record of all application activities;

vi. Network security: Next-generation firewalls (NGFW), intrusion detection system (IDS), intrusion prevention system (IPS), deep packet inspection (DPI) and secure protocols. General network layer protections;

vii. Secure development practices: code review, static (SAST) and dynamic (DAST) code analysis, vulnerability management for third-party components;

viii. Incident and response management: defining procedures and requirements, reporting and responding to incidents;

ix. Compliance: following standardization, regulations or laws that apply to the project;

x. Security training and awareness: users involved in the project must receive security training and awareness;

xi. Supply chain (Third-party): management of third-party assets and vulnerabilities;

a. Threat model: aims to understand the possible threats to the organization/project, the goal being to create a model to identify and mitigate vulnerabilities before the cybercriminal exploits them. The threat model involves the process of:

i. Asset identification: identifying which assets are critical to the company/project;

ii. Threat identification: identifying, enumerating and categorizing these assets with their risks (intentional or natural). Intentional attacks are understood to be targeted at the company or project, and natural attacks are understood to be natural disasters;

iii. Vulnerability assessment: identifying vulnerabilities that can be exploited by cybercriminals. Include design, architecture, and misconfiguration or default configuration;

iv. Risk analysis: prioritization of risk, which assets/applications need to be mitigated first;

v. Mitigation strategy: implementing and developing strategies that mature the security posture in terms of tools, processes and people;

vi. Review and update: adapting and continually reviewing the threat model.

2. Coding stage: requires secure and reliable development

a. Code analysis: fixing bugs, known vulnerabilities, licensing management and following best development practices;

i. Software composition analysis (SCA): identifies vulnerabilities and manages libraries, frameworks, dependencies and third-party components in open-source code:

1. Dependency scanning: catalogs and identifies third-party components and dependencies used in the project

2. Vulnerability detection: identifies known vulnerabilities in third-party components;

3. Licensing: verifies the licensing of third-party components used in the project, checking for possible legal impacts;

4. Risk management: carries out risk management and identifies severities (impacts);

5. Continuous monitoring: provides real-time information on newly discovered vulnerabilities or new versions (patches).

b. Static Application Security Testing (SAST): static technique used before deployment to statically analyze the code, looking for standardization, quality and security (known vulnerabilities):

i. Early bug detection: static analysis before deployment to identify possible bugs and code quality. Reduces cost and time effort;

ii. Process automation: configurations pre-establishing patterns, rules and heuristics to discover possible bugs. Making the process more efficient, scalable and faster;

iii. Identification of code vulnerabilities: identifies flaws in the code that could be exploited by cybercriminals, like SQL Injection (SQLi), XSS

iv. Code quality: static analysis can identify possible flaws, violations and writing validation (standardization);

v. Continuous integration: real-time feedback to the developer when they commit, integrated with the IDE.

c. Difference between SCA and SAST:

d. Dynamic Application Security Testing (DAST): a dynamic technique used after deploy stage, it analyzes the application's behavior (response) based on known real attacks. It is considered a black-box test because it simulates an external attack without knowing the architecture and source code:

i. Dynamic analysis: performs analysis in real time;

ii. Real attacks: simulates real attacks on the application;

iii. Vulnerabilities: identifies vulnerabilities, weak configurations that could be exploited and provides information to mitigate them;

iv. APIs and backend services: identifies vulnerabilities, weak configurations that could be exploited and provides information to mitigate them;

v. Reports: provides reports based on OWAS TOP 10, CIS, NIST etc.;

vi. Pipeline integration (CI/CD): blocks or notifies the system administrator of the vulnerability found, before deploying;

e. Interactive Application Security Testing (IAST): an interactive technique that analyzes and identifies vulnerabilities in real-time (runtime). Unlike SAST and DAST, which are performed manually or scheduled before and after deployment, IAST performs automatically through agents and sensors in real-time. It fills a gap that both [SAST and DAST] are unable to fill:

i. Real-time testing: monitors the application in real-time, analyzing behavior and detecting vulnerabilities;

ii. Deep analysis: analyzes the source code and the environment (application), offering insights through the interaction of different components;

iii. Low false positives: offers few false positives compared to DAST;

iv. Integration with CI/CD: integrates with pipelines, providing immediate feedback on security flaws;

v. Automation: security testing can be automated in the development flow;

f. Runtime Application Self-Protection (RASP): application self-protection technique. It automatically blocks malicious activities in real-time without any human interaction, i.e. any attempt at dubious or malicious behavior, such as executing a command (reverse shell) that harms the application or is not normally used, database injection (SQLi) or redirecting the page to a malicious external server (C2) — the service will be terminated/blocked. It is complemented by the Web Application Firewall (WAF), because if the attack goes beyond the perimeter zone, the RASP will be able to protect it. Zero-day attacks are reduced with this layer of protection:

i. Protection within the application: is integrated directly within the application (code and process), i.e. agents and sensors are installed to provide real-time response;

ii. Real-time threat detection: monitors application behavior in real time, detecting and responding to malicious activity. Activities considered malicious are blocked;

iii. Automatic response: activities considered malicious will be blocked automatically;

iv. Adaptive security policy: dynamically adapts the security policy — according to the application's behavior;

v. Extended protection: protects against common attacks such as SQLi, XSS, CSRF etc. It complements the WAF by analyzing request behavior and redirecting requests. RASP and WAF (together) can mitigate SSRF attacks;

vi. Visibility: provides insights and visibility only into the application's internal behaviors and environments;

vii. Integration with the DevOps process (lifecycle): can be integrated with the DevOps culture and procedural flow in CI/CD, pipelines, etc.;

g. Difference between IAST and RASP:

h. Code Review (best practices): helps maintain performance, quality and security in code maintenance like — SOLID, KISS, YAGNI, DRY, DDD and TDD.

3. Build stage: where the code is compiled, integrated and assembled into binaries and other artifacts. The main objective at this stage is to converge the language into binary, as well as resolving dependencies.

When it comes to secure development, it becomes a stage where tools must be deployed to check for vulnerabilities and countermeasures:

a. SAST

b. SCA

c. Detection of sensitive data in code (e.g. passwords)

d. DAST (analysis of vulnerabilities in APIs and container images)

4. Testing stage: various tests are carried out in order to guarantee the performance, quality, functionality, credibility and security of the application — before deploying it into production:

a. Unit testing: isolated testing of each component, checking its behavior and results (when necessary);

b. Integration testing: testing in conjunction with other integrations to check their behavior and results;

c. Functional testing: validates the application (Q&A);

d. Regression testing: ensures that new changes and functionalities do not break the application (like Content Security Policy [CSP]);

e. Performance testing: analyzes the application's responsiveness and scalability, including stress testing, load balancing, network bottlenecks, etc.;

f. Security testing: identifies vulnerabilities and potential security risks;

i. IAST (fitted into this stage);

g. User Acceptance Testing (UAT): end users assess whether the application meets the security standards and requirements agreed at the start of the project. This part is extremely important for validating the security features in the pipeline:

i. Security requirements: aspects agreed between the parties (access control, authentication mechanisms, data storage, type of encryption, compliance, etc.);

ii. Automated testing: reduces human risks and speeds up the validation process;

iii. Joint testing: Q&A should test the application from end to end — cloud environments (VMs, LBs, NoSQL, SQL, etc.), overloads, rollback, failover, bugs, APIs, data return, etc. It is important to comply with item 'ii', as automation helps to reduce human error;

iv. Continuous monitoring: detection of anomalies and failures, potential risks and vulnerabilities, metrics, etc. It is important to generate a weekly report to discuss with the technical team and metrics to track performance;

v. Awareness training: in this stage you should train those involved in the project to raise awareness about cybersecurity and phishing campaigns;

vi. Incident response planning and testing: plan, document and deploy a GRC (Governance, Risk and Compliance) and/or CIRT/CSIRT (incident response specialist team). Simulate an incident and recover the system, Mean Time Between Recovery (MTBR);

vii. Feedback: an articulation meeting to assess the positive and negative points between the departments involved in the project;

viii. Documentation: provide complete documentation and review it whenever necessary.

5. Release stage: the aim is to automate the release process for deployment, as well as to optimize and guarantee efficiency for the next stage, which is delivery:

a. Security automation: control and verification of configurations, patches, updates (latest version available) in all environments;

b. Versioning control (Azure DevOps):

i. Access control: based on RBAC by least privilege and conditional access;

ii. Authentication: MFA enabled;

iii. Logs: PAM (privileged access management) and continuous monitoring of privileged users (administrators);

iv. Appropriate Branch permissions: restrict ‘push’, ‘commits’, ‘pull requests’, ‘pull’, ‘merge’ etc. — on critical branches;

v. Digital signature: to verify the authenticity and integrity of commits, ensuring that there is no tampering;

vi. Backups: backup planning (internal/external) and recovery testing;

vii. Dependency update: latest available (stable) version;

viii. Default configuration: avoid the default configuration;

ix. Version strategy: identify exactly which versions have been released, for example — major. Sub major feature. Patch. (critical), i.e. 4.2.17.9.(1);

x. Segmentation: organize and separate folders into front-end, back-end, middleware, integrations, etc. Do not leave everything in a single folder (source) to facilitate auditing and security policies;

xi. Documentation: technical documentation of all the steps;

c. CI: Continuous Integration provides functional consistency of code across multiple contributors for automated delivery to the next layer;

d. Infrastructure as Code (IaC): manages and provisions cloud resources (infrastructure):

i. Consistency in environments: reduces risks of misconfiguration and gaps;

ii. Versioning and auditing: the configurations are stored in a file with the changes (versioning), making it possible to audit them;

iii. Reproducibility: reproduces the settings declared in the file in various environments (profile) and accounts distributed across the CSP, speeding up and automating;

iv. Mandatory policy: forces security policy to be deployed in the various distributed environments via a single script, and can work with profiles. It ensures that policies are applied correctly when provisioning, avoiding manual configuration errors. These policies include the security group (SG), allow logical ports, routing, asymmetric keys, type of encryption, etc;

v. Automated security test: before provisioning, a security test is carried out, looking for misconfigurations, incompatibilities, vulnerabilities, incorrect compliance, etc;

vi. Immutability: instead of maintaining the base system after an update, a new one is discarded and built, reducing surface attacks and vulnerabilities;

vii. Visibility: all the settings in the various environments are easily accessed via files, without the need to access each environment by environment;

viii. Centralization: manages all configurations in various cloud environments and accounts through a single platform;

6. Deliver stage: the crucial process of copying [signing] the code and infrastructure configurations and securely transferring [E2E] to the environment(s) — without tampering (MiTM):

a. Secure integration: if there are several cloud providers (CSP) involved in the deployment, sensitive information and passwords must be isolated and encrypted;

b. Monitoring: any failure in delivery must be notified to the system administrator.

7. Deployment stage: which the application becomes available to end users:

a. Validation: check that the application complies with the requirements and conformities;

b. Rollback: well-defined procedure for returning to the previous version — if necessary;

c. Continuous monitoring: record the entire deployment process (logs);

d. DAST.

8. Operate and Monitor stages: manages application environments, the aim is to guarantee availability, integrity:

a. SIEM: correlate and filter logs in different environments on a single platform;

b. SOAR: orchestrate and respond automatically to threats and failures;

c. Patch management: manage updates;

d. Scalability and Elasticity: provide dynamically resources (like vCPU, RAM, NVMe, LB, VMs, etc.);

e. Resilience: define the acceptable level of impact and implement mechanisms to help at this stage (e.g. failover, disaster recovery, etc.);

f. SOC: a specialized team monitoring the application (in real-time);

g. RASP

h. Application Performance Monitoring (APM): important for the user experience, it collects response metrics, resources used, packets trafficked, storage volume, overload notification, etc.

9. Feedback stage: proposes continuous communication with everyone involved in the project.

4. How does DevSecOps works?

DevSecOps is the secure integration of code through CI/CD tools. It follows a flowchart of pipeline timeline, covering software security checks throughout:

1. Code: The entire workflow starts from the root code to ensure static code analysis and code reviews are implemented in the coding phase for the syntax prone to security threats.

2. Commit: The commit made to the git repository needs to be passed through the right level of security by working in a private repository instead of the public repository to prevent any threat exposure. The CI pipeline starts after the Commit phase.

3. Build and Test: This is a combined phase of static code analysis identifying vulnerabilities, performing integration tests and performance tests along with infrastructure scans. This pipeline interval is called as CI pipeline.

4. Staging and Production: This phase of the pipeline is called a CD part of the pipeline and includes a review in staging and production with a parallel passive penetration test, and SSL scan to ensure the production-ready code is well protected.

5. Exline well known DevSecOps tools.

Tools are the efficient utility of the DevSecOps model that helps to fast-pace the software development environment. They are integrated into the DevOps pipeline. There are several tools used to ensure the safety of data and the implementation of security in software processes.

Tools are categorised into several genres like Code Analysis, Change Management, Compliance Monitoring, Threat Investigation and Vulnerability Management while integrating them

separately through different phases of SDLC. There are certain categories in which tools are been segregated to ensure secure application development:

1. Code Analysis

This category of DevSecOps demonstrates empowerment of security in the coding phase of development. Tools like SAST (Static Application Security Testing) and DAST (Dynamic Application Security Testing) ensure security and keep check of threat analysis in a given developer's source code with a predefined set of rules and patterns.

2. Change Management

This category represents any change or modification that happened during the application development. It helps in the continuous improvement of code and fixes potential vulnerabilities and changes.

Example:

Jenkins, Travis CI automates changes and integration to the development process.

3. Compliance Monitoring

There are certain tools which focus on compliance features such that the software composition analysis (SCA) automatically monitors future risk management and security compliance.

Example:

Nagios, Zabbix, and Splunk monitor the performance of the code.

4. Threat Investigation and Vulnerability Management

DevSecOps professionals use tools like Interactive Secure Application Testing (ISAT) to evaluate threats in the runtime environment of software development.

6. What are the benefits of DevSecOps?

These are some benefits of using devsecops :

It boosts the delivery system of applications in organizations and increases the efficiency of applications.

Continuous Quality Improvement:

DevSecOps is important for software development because it helps to keep people safe by making sure that the software is secure.

increased security

improved workflow efficiency

faster product releases

better customer experience

Reduce Vulnerabilities and Bugs Early On

Streamline Compliance Reporting

Save Costs on Resource Management

Make Developers Security-Aware

Reduce Risk and Legal Liability

Faster Delivery of Software

Proactive And Improved Overall Security

Automation Compatibility and Modern Development

7. About Local and international DevSecOps career opportunities, career path.

At this time security is one issue in the world and technologies become to enhance to security this approach or devsecops is career with knowledge of security. Devsecops engineers have special skills to mitigate the security issue.

So many careers opportunity like

API Security Engineer

Cloud DevSecOps Engineer

Development Security Operations Engineer

Devops And Automation Security Engineer

DevSecOps Analyst

DevSecOps And Site Reliability Engineer

DevSecOps Architect

DevSecOps Automation Engineer

DevSecOps CI/CD Engineer

DevSecOps Container Engineer

DevSecOps Engineer

DevSecOps Platform Engineer

DevSecOps Site Reliability Engineer

DevSecOps Testing Engineer

With the increased need for proper data security, there is no shortage of job security for someone working in DevSecOps. The opportunities are vast, including small businesses and large corporations across the country.

Remote and in-person opportunities exist, although the company's size will be a prime decider. DevSecOps work commonly involves interdepartmental communication across physical and digital mediums. The higher one gets in an organization's structure, the more likely it will become that they need to be physically present.

Teaching is just as much a part of this career as learning, and being in the most comfortable spot to deliver important information is advisable. No matter how excellent one's communication skills can be, in the wrong environment, it will not matter.

Working with other industry professionals is the standard, and can involve interacting with large workforces to optimize and explain large and complex concepts succinctly. Organizations require unified digital security whether they are aware of it or not. The demand for this specialization is only increasing, which means more potential places of employment. Steps in the Devsecops Lifecycle

DevSecOps is a software development methodology that emphasizes security and collaboration between development, security, and operations teams throughout the software development lifecycle. DevSecOps works best with teams that use CI/CD, or continuous integration and delivery process, meaning code changes are integrated and released as part of an automated process.

The DevSecOps lifecycle can be broken down into the following steps, with the development, testing, and deployment stages often happening in a loop as software updates are made and new features are added:

DevSecOps Engineers: These individuals are responsible for the configuration of the IT structure, identifying security threats, and securing software development. Their job is very similar to that of a good deal of IT security professional roles.

References:

- geeksforgeeks.com
- Javatpoint.com
- Linkedin.com