Notion de risques : Chap.6.a : Sécurité des logiciels

lundi 9 décembre 2024 19:39

Chapitre 6.a : Sécurisation des Environnements dans le Contexte de SDLC et DevSecOps

Objectifs

- Aborder les méthodes de développements
- Aborder les concepts de sécurisation des environnements
- Aborder les processus et les outils associés aux environnements
- Proposer des processus et des outils pour l'environnement Web N1, N2 et N3

A. Introduction

La sécurité des logiciels est essentielle pour protéger les systèmes d'information et les données contre les attaques. Avec l'augmentation des cybermenaces, il est crucial d'intégrer des pratiques de sécurité tout au long du cycle de vie du développement logiciel (SDLC) ou DevSecOps.
Ce chapitre explore les modèles de sécurisation, l'imbrication des modèles de développement et de sécurisation, les processus et les outils associés, pour terminer une proposition pour un environnement WEB de développement, d'intégration et de production.

B. Concepts de sécurisation des environnements de développement

B.1 : Les modèles de sécurisation

- **DevSecOps :**
  - Le DevSecOps est un modèle qui englobe le développement (Dev), la sécurité (Sec) et l'exploitation (Ops). Cette approche de la culture, de l'automatisation et de la conception des plateformes intègre la sécurité en tant que responsabilité partagée tout au long du cycle de vie informatique.
    - Sources : DevSecOps | ANSSI
    - Règles de durcissement sur les OS hébergeant les applications (cf. https://cyber.gouv.fr/guide-linux).
- **SDLC : Secure DevLife Cycle :**
  - Le **Secure Software Development Life Cycle (SDLC)** ou cycle de vie du développement logiciel sécurisé est un processus qui intègre la sécurité dans chaque phase du développement logiciel. Cela inclut la planification, la conception, le codage, les tests, le déploiement et la maintenance des logiciels. L'idée est de prévoir la sécurité dès les premières étapes de la conception, d'effectuer des évaluations des risques, d'assurer un codage sécurisé, et de tester régulièrement les vulnérabilités.
- **SBOM : Software Bill of Material :**
  - Le **Software Bill of Material (SBOM)** est un inventaire complet des composants logiciels d'un produit. Il fournit une liste détaillée de toutes les bibliothèques, frameworks, et autres composants tiers utilisés dans une application, ce qui permet de mieux comprendre les dépendances et d'identifier plus rapidement les vulnérabilités. L'usage d'un SBOM facilite la gestion de la sécurité et la conformité en assurant la transparence des composants logiciels.

B.2 : L'imbrication des modèles Agile, DevOps, DevSecOps, SDLC

1) Les méthodes de développement : Agile et DevOps

- **Méthode Agile** :
  - **Objectif :** L'Agilité repose sur des cycles de développement courts, appelés **sprints**, permettant une livraison rapide de fonctionnalités. Elle favorise la collaboration continue entre le **Product Owner**, les **développeurs** et les **parties prenantes**.
  - **Processus** :
    - La **planification** se fait de manière itérative (**Sprint Planning**), chaque fonctionnalité étant définie comme une **User Story**, elle s'ajoute à un **BackLog**.
    - Les **tests unitaires** sont intégrés au code dès le début, favorisant une évolution rapide et continue du produit.
    - **Outils associés :** Jira, Trello pour le suivi des tâches, Confluence pour la documentation.
  - **Processus à suivre pour ajouter une fonctionnalité**
    - **Etape 0 : Kickoff**
    - Étape 1 : Définir la fonctionnalité dans le Product Backlog
    - Étape 2 : Planification du Sprint
    - Étape 3 : Développement et Implémentation
    - Étape 4 : Tests et validation (TDD: Test Driven Developpment)
    - Étape 5 : Revue du Sprint (Sprint Review)
    - Étape 6 : Rétrospective du Sprint (Sprint Retrospective)
    - **Etape 7 : Mise en Production**

- **Méthode DevOps** :
  - **Objectif :** DevOps vise à rapprocher les équipes de développement (Dev) et des opérations (Ops) pour favoriser un déploiement continu et fiable en production.
  - **Processus :**
    - Mise en place de l'**intégration continue (CI)** et de la **livraison continue (CD)**.
    - Déploiement automatisé avec des **tests automatisés** et une surveillance en temps réel pour garantir la stabilité du système.
    - **Outils associés :** Jenkins, GitLab CI/CD, Docker pour la gestion des conteneurs, Kubernetes pour l'orchestration.
  - **Processus à suivre…**
    - **Lancement du projet (Planification et définition des objectifs)**
    - Gestion du code source (Versioning et gestion des branches)
    - Développement (Cycle de développement agile)
    - Tests continus et intégration continue (CI)
    - Livraison continue (CD) et automatisation des déploiements
    - Surveillance et gestion de la performance en production
    - Gestion des incidents et retour d'information

- **Différences principales entre Agile et DevOps**
  - Objectifs
    - **Agile :** Se concentre sur le développement logiciel, la gestion des projets et la fourniture de fonctionnalités de manière rapide et itérative. L'objectif est de produire un produit fonctionnel à la fin de chaque sprint tout en favorisant la collaboration avec le client.
    - **DevOps :** Se concentre sur la collaboration entre les équipes de développement et d'exploitation pour améliorer le processus global de livraison et de gestion des applications. DevOps vise à rendre les déploiements plus rapides, plus fiables et à renforcer la surveillance continue en production.
  - Focus de la méthode
    - **Agile :** Agile se concentre sur **le développement et la livraison des fonctionnalités logicielles**. Cela implique des cycles de développement plus courts, une gestion flexible des exigences, et des ajustements constants basés sur les retours clients.
    - **DevOps :** DevOps se concentre sur **l'intégration et la livraison continues** du logiciel tout en assurant sa stabilité et sa sécurité en production. L'objectif est d'automatiser le déploiement, le test et l'intégration pour augmenter la fréquence des mises à jour.

- Cycle de vie du projet
  - **Agile :** Agile est principalement concerné par le **développement** du produit et la gestion du cycle de vie des fonctionnalités. Il se concentre sur la phase de développement, les interactions avec les utilisateurs finaux et la flexibilité dans l'adaptation du produit aux besoins changeants.
  - **DevOps :**DevOps couvre l'intégralité du **cycle de vie du logiciel**, y compris la planification, le développement, les tests, le déploiement, la surveillance et la gestion des opérations en production. Il intègre des pratiques pour l'automatisation du développement et des opérations.
- Rôles et Collaboration
  - **Agile :** Les équipes **Agiles** sont souvent composées de développeurs, de testeurs et de parties prenantes métier (comme les Product Owners), qui collaborent de manière itérative pour définir et créer des fonctionnalités. La communication avec le client ou le propriétaire du produit est au centre du processus Agile.
  - **DevOps : DevOps** favorise la collaboration entre **les équipes de développement (Dev)** et **les équipes d'exploitation (Ops)**. L'objectif est d'éliminer le fossé traditionnel entre ces deux équipes, qui travaillaient généralement de manière indépendante, en assurant une collaboration continue pour l'intégration et le déploiement rapides des applications.
- Automatisation
  - **Agile :** Bien que Agile valorise l'automatisation des tests (par exemple avec **Test-Driven Development (TDD)** ou **Continuous Testing**), elle n'inclut pas d'automatisation dans le déploiement et la gestion des environnements, qui restent des responsabilités distinctes.
  - **DevOps :** DevOps met un accent fort sur l'**automatisation** à travers **CI/CD**, **Infrastructure as Code (IaC)**, et des tests automatisés, pour garantir des déploiements continus et sûrs. L'automatisation dans DevOps touche non seulement le développement, mais aussi les environnements de test, de staging et de production.
- Mesure du succès
  - **Agile :** Le succès est mesuré par la capacité à livrer des fonctionnalités de qualité qui répondent aux besoins des utilisateurs, en fonction des itérations et des retours obtenus à chaque sprint.
  - **DevOps :** Le succès est mesuré par la rapidité, la fréquence et la fiabilité des déploiements en production, ainsi que par la stabilité et la performance des applications dans des environnements de production en continu.

2) Les concepts de sécurisation des environnements avec SDLC, DevSecOps

- **SDLC (Secure Software Development Life Cycle)** :
  - **Objectif :** Intégrer la sécurité à chaque étape du cycle de vie du développement logiciel. SDLC inclut des évaluations de sécurité dès la phase de conception et la mise en place de tests de sécurité tout au long du développement.
  - **Processus :**
    - Chaque phase (planification, conception, développement, tests) prend en compte les aspects sécuritaires.
    - **Tests de vulnérabilités**, **gestion des configurations** et **audits réguliers**.
    - **Outils associés :** SonarQube pour l'analyse du code, Fortify pour les tests de sécurité, Checkmarx pour la gestion des vulnérabilités.
  - **Processus à suivre…**
    - Planification et définition des exigences
    - Conception (Design & Security)
    - Développement et programmation (avec un focus Security)
    - Tests de sécurité
    - Déploiement et mise en production
    - Surveillance et maintenance post-production

- **DevSecOps** :
  - **Objectif :** DevSecOps intègre la sécurité dans les pratiques de DevOps, c'est-à-dire l'automatisation des tests de sécurité dans le pipeline CI/CD, de manière continue et proactive.
  - **Processus :**
    - **Automatisation des tests de sécurité** à chaque étape du développement et du déploiement.
    - Collaboration continue entre **les équipes de développement, d'opérations et de sécurité**.
    - **Outils associés :** Jenkins pour l'intégration continue avec des plugins de sécurité, HashiCorp Vault pour la gestion des secrets, OWASP ZAP pour les tests de sécurité dynamique.
  - **Processus à suivre…(+ ajout du processus sécurité)**
    - **Lancement du projet (Planification et définition des objectifs)**
    - Gestion du code source (Versioning et gestion des branches) + gestion de la sécurité dans le code
    - Développement (Cycle de développement agile) + tests de sécurité intégrés
    - Tests continus et intégration continue (CI) + validation des vulnérabilités
    - Livraison continue (CD) + automatisation des déploiements sécurisés
    - Surveillance et gestion de la performance en production + focus sécurité
    - Gestion des incidents et retour d'information + focus sécurité
    - + Gestion des mises à jour et des correctifs de sécurité
    - + Révision et rétroaction (Rétrospective DevSecOps)

- **Différences principales entre SDLC et DevSecOps**
  - **SDLC (Secure Software Development Life Cycle)**
    - Le **SDLC** fait référence au processus global de développement logiciel, de sa conception initiale à son déploiement et sa maintenance.
    - Le **Secure SDLC** (ou **SDLC sécurisé**) est un cadre qui intègre des pratiques de sécurité à chaque étape du développement logiciel. Cela inclut des actions comme l'analyse des risques, l'intégration de contrôles de sécurité dès la phase de conception, l'audit de sécurité du code, et des tests de vulnérabilité à chaque étape du cycle de vie.
    - Le SDLC met l'accent sur la planification, la conception, le développement, les tests, le déploiement et la maintenance sécurisés des logiciels.
  - **DevSecOps**
    - **DevSecOps** (Development, Security, and Operations) est une approche qui intègre la sécurité dès les premières étapes du développement logiciel, mais aussi tout au long de l'exploitation et des opérations du logiciel. Contrairement à un SDLC classique où la sécurité peut être traitée à la fin du développement ou par une équipe dédiée, **DevSecOps** met l'accent sur la **sécurisation continue et l'automatisation** de la sécurité tout au long du cycle de vie du logiciel.
    - **DevSecOps** vise à automatiser les tests de sécurité, l'intégration continue et la surveillance, ce qui permet aux équipes de développement et d'exploitation de travailler plus rapidement tout en minimisant les risques liés à la sécurité.
    - L'objectif est de faire en sorte que la sécurité soit intégrée **dès le début**, en utilisant des outils de sécurité automatisés, des tests de pénétration en continu et des audits de sécurité intégrés dans les pipelines CI/CD (intégration continue / déploiement continu).

- **En résumé :**
  - **SDLC** se concentre sur l'intégration de la sécurité dans chaque phase du développement logiciel.
  - **DevSecOps** est un modèle plus large et plus intégré qui combine développement, sécurité et opérations, avec une approche de sécurité automatisée et continue.

3) Les processus et les outils associés aux environnements

- **a) Lors de l'utilisation de SDLC** :
  - Dans le cadre de SDLC, chaque étape du développement est soigneusement contrôlée pour s'assurer que la sécurité est respectée.
  - Les tests de sécurité sont effectués régulièrement au cours du cycle de développement, notamment lors de la phase de test et de déploiement.
  - **Outils associés à SDLC** :
    - **Git** pour le contrôle de version, **SonarQube** pour la gestion de la qualité du code, **OWASP Dependency-Check** pour vérifier les vulnérabilités dans les dépendances, et **Jenkins** pour l'automatisation du pipeline de tests.
- **b) Lors de l'utilisation de DevSecOps** :
  - L'intégration continue est associée à des tests de sécurité automatisés à chaque modification de code, garantissant ainsi la détection immédiate des vulnérabilités.
  - DevSecOps met l'accent sur des pratiques de sécurité **en continu** tout au long du cycle de vie du logiciel.
  - **Outils associés à DevSecOps** :
    - **Jenkins** ou **GitLab CI/CD** pour l'intégration continue.
    - **Snyk** pour analyser les dépendances open source et leur sécurité, **OWASP ZAP** et **Burp Suite** pour les tests de sécurité dynamiques.

4) Articulation des trois concepts dans le projet avec 3 environnements : Développement, Test/Intégration, et Production

a) Environnement de Développement (Dev)

L'environnement de développement est l'endroit où les développeurs écrivent le code. L'objectif est de travailler sur de nouvelles fonctionnalités de manière rapide et continue, en respectant les principes **Agile**.

- **Processus :**
  - Le **Product Owner** définit les **User Stories** et les priorités dans le **Backlog**.
  - Les **développeurs** créent des branches pour les nouvelles fonctionnalités et travaillent dans un environnement isolé.
  - **Tests unitaires** et **revues de code** régulières.
  - **Intégration continue** pour vérifier l'intégrité du code.
- **Outils associés :**
  - **IDE** : PHPStorm, Visual Studio Code.
  - **VCS** : **Git** avec **GitHub**, **GitLab** ou **Bitbucket** pour la gestion du code source.
  - **Tests unitaires** : PHPUnit pour PHP.
  - **CI** : **Jenkins**, **GitLab CI** ou **Travis CI** pour l'intégration continue.
  - **Docker** : Pour simuler des environnements de développement isolés (environnements conteneurisés).

b) Environnement de Test et d'Intégration (Test/Integration)

Cet environnement est utilisé pour tester les nouvelles fonctionnalités développées et pour intégrer le code des différents développeurs. Il est également utilisé pour tester la sécurité avant le déploiement en production.

- **Processus :**
  - **Tests d'intégration** pour vérifier que toutes les fonctionnalités s'intègrent correctement entre elles.
  - **Tests de régression** pour garantir qu'aucune fonctionnalité existante n'est cassée par les modifications.
  - **Tests de sécurité** automatiques dans le pipeline CI/CD, y compris des analyses statiques du code, des tests de pénétration et des audits de vulnérabilités.
  - **Contrôles de qualité** pour vérifier la conformité du code aux normes de sécurité et de performance.

- **Outils associés :**
  - **CI/CD** : **Jenkins**, **GitLab CI/CD**.
  - **SAST** (Static Application Security Testing) : **SonarQube**, **Checkmarx** pour analyser la sécurité du code source.
  - **DAST** (Dynamic Application Security Testing) : **OWASP ZAP**, **Burp Suite** pour tester les vulnérabilités en dynamique.
  - **Tests automatisés** : **Selenium**, **Cypress** pour tester les interfaces utilisateur.

c) Environnement de Production (Prod)

C'est l'environnement où l'application est déployée et où les utilisateurs finaux interagissent avec l'application. L'objectif est de maintenir la stabilité et la sécurité en production.

- **Processus :**
  - **Déploiement continu** en production à l'aide de pipelines CI/CD. Le code validé dans les environnements de test est déployé automatiquement en production.
  - **Tests de sécurité en production** : Effectuer une surveillance continue des vulnérabilités et des failles en production.
  - **Gestion des secrets** et des accès via des outils de **gestion des configurations** et des **secrets**.
  - **Monitoring** pour détecter les incidents de sécurité et de performance en temps réel.
- **Outils associés :**
  - **Docker**, **Kubernetes** pour la gestion des conteneurs et l'orchestration des microservices en production.
  - **Vault** de **HashiCorp** pour la gestion sécurisée des secrets.
  - **Prometheus**, **Grafana** pour la surveillance des métriques de performance en temps réel.
  - **Datadog**, **New Relic** pour la surveillance et la gestion des incidents de sécurité en production.

5) Outils et processus associés pour ce projet

Voici un récapitulatif des outils et des processus associés pour la méthode **SDLC**, **Agile**, et **DevSecOps** dans un projet de développement en PHP/MySQL avec les 3 environnements :

1. Gestion des User Stories et du Backlog :
- **Outils** : **Jira** (Agile), **Trello** (Agile), **Confluence** (documentation du backlog et des exigences).

2. Développement et gestion de code :
- **Outils** : **Git**, **GitHub** ou **GitLab** pour le contrôle de version, gestion des branches.
- **CI/CD** : **Jenkins**, **GitLab CI**, ou **Travis CI** pour l'intégration continue et les tests.

3. Tests et validation :
- **Outils de test** : **PHPUnit** (tests unitaires), **Selenium**, **Cypress** (tests fonctionnels).
- **Outils de sécurité** : **OWASP ZAP**, **Burp Suite**, **SonarQube** pour l'analyse statique et dynamique du code.

4. Déploiement et sécurité :
- **Outils de gestion de secrets** : **Vault** de HashiCorp.
- **Docker** et **Kubernetes** pour la gestion des environnements de production et de test.
- **Prometheus** et **Grafana** pour la surveillance de la performance et de la sécurité en production.

5. Monitoring et gestion des incidents :
- **Outils de surveillance** : **Datadog**, **New Relic**, **Prometheus**, **Grafana** pour la gestion de la performance et la détection des incidents.

B.3 : Software Bill of Materials

- **Objectif :** Le **SBOM** (Software Bill of Materials) permet de lister tous les composants logiciels utilisés dans une application, y compris leurs versions, licences et vulnérabilités connues, afin de garantir la transparence et la gestion des risques liés aux dépendances logicielles.

- **Processus :**
  a. **Identification des composants** : Identifier toutes les dépendances tierces (bibliothèques, frameworks) utilisées dans le développement.
  b. **Création et gestion du SBOM** : Générer et maintenir à jour un fichier SBOM, qui documente toutes les dépendances et leurs versions.
  c. **Tests et gestion des vulnérabilités** : Analyser les vulnérabilités des composants listés dans le SBOM à chaque étape du développement.
  d. **Mise à jour en production** : Assurer que le SBOM reste à jour tout au long du cycle de vie de l'application, y compris en production.

- **Processus à suivre… :**
  a. **Lors de la planification** : Identifier et choisir les bibliothèques nécessaires.
  b. **Lors du développement** : Générer et mettre à jour le SBOM au fur et à mesure de l'intégration de nouvelles dépendances.
  c. **Lors des tests** : Effectuer une analyse de sécurité (SAST, DAST) sur les composants listés dans le SBOM.
  d. **Lors du déploiement** : Mettre à jour le SBOM et s'assurer qu'aucune vulnérabilité n'est présente dans les composants en production.

- **Les outils :**
  - **CycloneDX**, **SPDX** : Outils de génération et de gestion du SBOM.
  - **Snyk**, **OWASP Dependency-Check** : Pour analyser les vulnérabilités des dépendances.
  - **Jenkins**, **GitLab CI/CD** : Pour automatiser les tests et le déploiement avec des contrôles de sécurité.

C. Référence

C.1 : Références pour Développer la Curiosité (DevSecOps)

Pour approfondir la compréhension de DevSecOps, voici quelques références et ressources clés :

- **OWASP DevSecOps** : OWASP DevSecOps Project propose des pratiques et des outils pour intégrer la sécurité dans DevOps.
- **Book** : *"The DevOps Handbook"* par Gene Kim, Patrick Debois, John Willis et Jez Humble, qui présente une approche de la sécurité dans le cadre DevOps.
- **Articles et Blogs** : Le blog de **DevOps.com** propose de nombreux articles sur l'intégration de la sécurité dans DevOps et DevSecOps.
- **Certifications** : Des certifications comme la **Certified DevSecOps Professional (CDP)** ou **Certified Kubernetes Security Specialist (CKS)** sont des ressources intéressantes pour approfondir le sujet.

C.2 : Références pour Développer la Curiosité (**SBOM)**

Pour approfondir la compréhension de SBOM, voici quelques références et ressources clés :

- **CycloneDX** : Site officiel
- **OWASP SBOM** : OWASP SBOM Documentation
- **Snyk** : Snyk - Secure your open source