UNIVERSITÉ DU QUÉBEC

CESI
CENTRE D'EXPERTISE EN SÉCURITÉ DE L'INFORMATION

TLP : VERT (DIFFUSION PERMISE)

Test de sécurité applicative (SAST, DAST et SCA)

09-2024

UNIVERSITÉ DU QUÉBEC

CESI
CENTRE D'EXPERTISE EN SÉCURITÉ DE L'INFORMATION

TLP : VERT (DIFFUSION PERMISE)

TABLE DES MATIÈRES

INTRODUCTION

IMPORTANCE DES TESTS DE SÉCURITÉ APPLICATIVE FACE AUX CYBERMENACES

Dans un contexte où les cybermenaces deviennent de plus en plus sophistiquées et fréquentes, la sécurité des applications est devenue une priorité incontournable pour les organisations. Les applications modernes sont souvent composées de plusieurs couches, incluant du code propriétaire, des composants tiers et des bibliothèques « open source ». Chacune de ces couches peut introduire des vulnérabilités[1] qui, si elles ne sont pas correctement gérées, peuvent être exploitées par des attaquants.

PRÉSENTATION DES TESTS DE SÉCURITÉ SAST, DAST ET SCA

Les tests de sécurité statique des applications (SAST), les tests de sécurité dynamique des applications (DAST) et l'analyse de la composition logicielle (SCA) sont trois approches complémentaires pour tester la sécurité des applications. Le SAST analyse le code source pour identifier les vulnérabilités potentielles sans exécuter l'application, tandis que le DAST teste l'application en cours d'exécution pour détecter les failles de sécurité et le SCA examine les composants tiers et les bibliothèques « open source » pour déceler les vulnérabilités connues. Ensemble, ces méthodes offrent une couverture complète des tests de sécurité, permettant de détecter et de corriger un large éventail de vulnérabilités.

TESTS DE SÉCURITÉ STATIQUE DES APPLICATIONS (SAST)

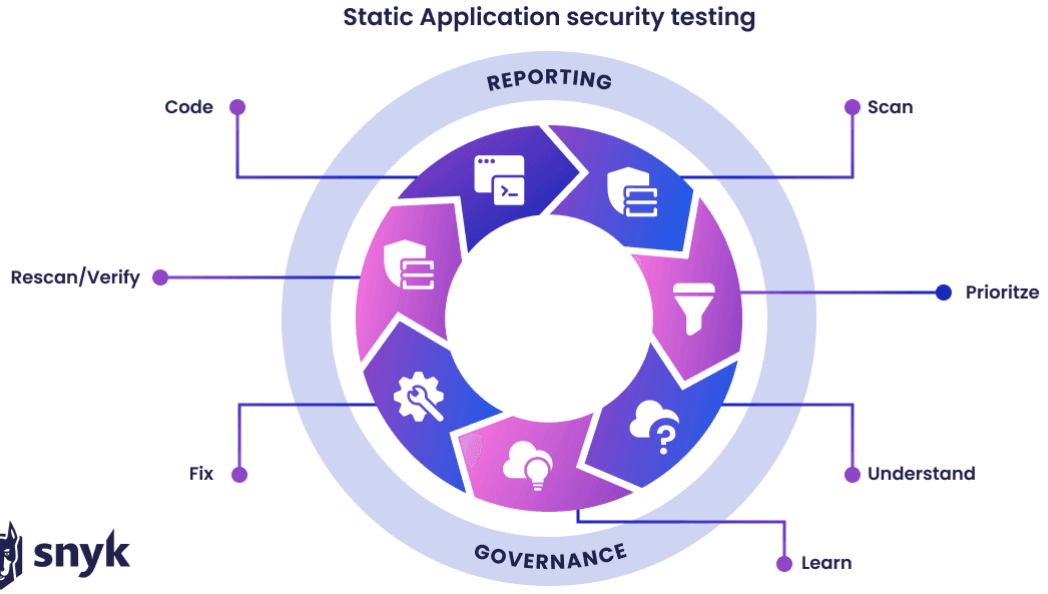
QU'EST-CE QUE LE SAST?

Le SAST est une méthode de test de sécurité qui analyse le code source ou le « bytecode » d'une application pour détecter des vulnérabilités potentielles. Ces analyses se font sans exécuter le programme, permettant de trouver des failles dès les premières étapes du développement. Le SAST est particulièrement efficace pour identifier les erreurs de codage, les failles de conception et les mauvaises pratiques de programmation.

[1] Vulnérabilité : Une faiblesse dans un système, une application ou un réseau qui peut être exploitée par une menace pour violer les politiques de sécurité.

LES ÉTAPES DES TESTS DE SÉCURITÉ SAST

1. **Sélection des outils :** Choisir des outils SAST adaptés aux technologies utilisées.
2. **Analyse initiale :** Effectuer une première analyse du code pour identifier les vulnérabilités potentielles.
3. **Révision des résultats :** Examiner les résultats pour éliminer les faux positifs[2] et prioriser les vulnérabilités à corriger.
4. **Correction des vulnérabilités :** Apporter les modifications nécessaires au code pour corriger les vulnérabilités identifiées.
5. **Réanalyse :** Effectuer une nouvelle analyse pour vérifier que les corrections ont été efficaces et qu'aucune nouvelle vulnérabilité n'a été introduite.



Source : Les 7 étapes des tests de sécurité des applications statiques (SAST) – Snyk

[2] Faux positif : Un résultat de test qui indique à tort la présence d'une vulnérabilité alors qu'il n'y en a pas.

FONCTIONNEMENT DES OUTILS SAST

Les outils SAST fonctionnent en effectuant un balayage sur le code source et en utilisant des règles prédéfinies pour identifier les vulnérabilités. Ils détectent des problèmes comme les injections SQL[3], les XSS[4] et les mauvaises pratiques de codage. Certains outils utilisent des techniques d'analyse statique avancées, telles que l'analyse de flux de données et l'analyse de contrôle de flux, pour identifier des vulnérabilités complexes.

AVANTAGES ET INCONVÉNIENTS DES TESTS SAST

AVANTAGES

- Identification précoce des vulnérabilités.
- Analyse complète du code.
- Intégration facile dans le processus de développement.

INCONVÉNIENTS

- Faux positifs fréquents.
- Nécessite une connaissance approfondie du code.
- Ne détecte pas les vulnérabilités qui apparaissent seulement à l'exécution.

CAS D'UTILISATION COURANTS DU SAST

Le SAST est souvent utilisé dans les phases initiales du développement, lors des revues de code et dans les processus de CI/CD[5], pour assurer une vérification continue du code. Il est particulièrement utile pour les applications critiques, où la sécurité du code source est primordiale.

[3] Injection SQL (SQLi) : Une vulnérabilité qui permet à un attaquant d'injecter des requêtes SQL malveillantes dans une application, généralement via des champs de saisie utilisateur, pour manipuler la base de données sous-jacente.
[4] Cross-Site Scripting (XSS) : Une vulnérabilité où un attaquant injecte du code malveillant (souvent du JavaScript) dans une application web que d'autres utilisateurs exécutent sans le savoir.
[5] CI/CD (Continuous Integration/Continuous Deployment) : Ensemble de pratiques de développement logiciel où les modifications de code sont automatiquement testées, intégrées et déployées dans un environnement de production. Le CI/CD permet un déploiement rapide et fiable des applications tout en assurant la qualité grâce à des tests automatisés.

INTÉGRATION DU SAST DANS LE CYCLE DE VIE DU DÉVELOPPEMENT LOGICIEL (SDLC)

L'intégration du SAST dans le SDLC permet de détecter les vulnérabilités à chaque étape du développement, de la conception à la maintenance, garantissant ainsi une sécurité continue. Les outils SAST peuvent être intégrés dans les environnements de développement intégrés (IDE) et les pipelines CI/CD pour automatiser les analyses de sécurité.

VULNÉRABILITÉS DÉTECTÉES PAR LE SAST ET LEUR RELATION AVEC L'OWASP TOP 10

Rang	Catégorie OWASP Top 10 2021	Description	Relation avec SAST
A01	Bris d'accès (Broken Access Control)	Failles dans les mécanismes de contrôle d'accès, permettant à des attaquants de contourner les autorisations.	Potentiel
A02	Exposition de données sensibles (Sensitive Data Exposure)	Mauvaise gestion de la protection des données sensibles, comme les informations personnelles ou financières.	**Oui**
A03	Injection	Failles liées à l'injection de code non contrôlé dans des requêtes SQL, LDAP, XPath, ou des systèmes similaires.	**Oui**
A04	Conception peu sûre (Insecure Design)	Problèmes architecturaux liés à une conception de sécurité insuffisante.	Non
A05	Erreurs de configuration de sécurité (Security Misconfiguration)	Défauts dans les configurations qui exposent les systèmes à des attaques.	Potentiel
A06	Utilisation de composants présentant des vulnérabilités connues (Vulnerable and Outdated Components)	Risques liés à l'utilisation de bibliothèques, frameworks ou composants logiciels vulnérables ou obsolètes.	Non
A07	Identification et authentification insuffisantes (Identification and Authentication Failures)	Problèmes dans les mécanismes d'authentification, permettant des accès non autorisés.	**Oui**
A08	Échec de l'intégrité des logiciels et des données (Software and Data Integrity Failures)	Problèmes liés à la falsification de logiciels ou de données pendant les	Non

		processus de mise à jour ou de déploiement.	
A09	Insuffisance des journaux et de la surveillance (Security Logging and Monitoring Failures)	Absence de surveillance et de journalisation suffisante pour détecter et réagir aux incidents de sécurité.	Non
A10	Défaillances du contrôle côté serveur (Server-Side Request Forgery - SSRF)	Failles permettant à un attaquant d'envoyer des requêtes malveillantes à un serveur tiers.	Non

TESTS DE SÉCURITÉ DYNAMIQUE DES APPLICATIONS (DAST)

QU'EST-CE QUE LE DAST?

Le DAST est une méthode de test de sécurité qui analyse une application en cours d'exécution pour détecter des vulnérabilités. Contrairement au SAST, le DAST teste l'application depuis l'extérieur, en simulant les attaques potentielles. Cette approche permet d'identifier les vulnérabilités liées à l'exécution, telles que les failles d'injection, les erreurs de configuration et les problèmes de logique applicative.

LES ÉTAPES DES TESTS DE SÉCURITÉ DAST

1. **Définition de la portée :** Déterminer les composants de l'application à tester (Ex. : API, interfaces utilisateur).
2. **Configuration des outils :** Configurer les outils DAST pour interagir avec l'application cible.
3. **Exécution des tests :** Lancer des tests automatiques et manuels pour identifier les vulnérabilités.
4. **Analyse des résultats :** Examiner les résultats pour identifier les vulnérabilités et les faux positifs.
5. **Correction des vulnérabilités :** Apporter les modifications nécessaires à l'application pour corriger les vulnérabilités identifiées.
6. **Réexécution des tests :** Effectuer de nouveaux tests pour vérifier l'efficacité des corrections et détecter d'éventuelles nouvelles vulnérabilités.

FONCTIONNEMENT DES OUTILS DAST

Les outils DAST interagissent avec l'application via son interface utilisateur, ses API ou ses points de terminaison (endpoints) web. Ils envoient des requêtes malveillantes pour identifier les failles de sécurité comme les injections SQL, les XSS et les problèmes de configuration. Certains outils DAST peuvent également effectuer des analyses de « fuzzing »[6] pour détecter des vulnérabilités inconnues.

AVANTAGES ET INCONVÉNIENTS DES TESTS DAST

AVANTAGES :

- Détection des vulnérabilités en conditions réelles.
- Test des configurations de sécurité et des environnements de production.
- Identification des problèmes que le SAST ne peut pas trouver.

INCONVÉNIENTS :

- Ne détecte pas les vulnérabilités dans le code non exécuté.
- Nécessite un environnement complet pour les tests.
- Moins efficace, pour identifier les vulnérabilités internes du code.

CAS D'UTILISATION COURANTS DU DAST

Le DAST est souvent utilisé en phase de test et de production pour s'assurer que l'application est sécurisée une fois déployée. Il est également utilisé pour les tests de pénétration et les audits de sécurité. Le DAST est particulièrement utile pour tester les applications web et les API, où les interactions externes sont fréquentes.

INTÉGRATION DU DAST DANS LE CYCLE DE VIE DU DÉVELOPPEMENT LOGICIEL SDLC

L'intégration du DAST dans le SDLC permet de tester la sécurité des applications à différentes étapes, notamment avant la mise en production, assurant que les applications sont sécurisées

[6] Fuzzing : Une technique de test qui envoie des entrées aléatoires à une application pour provoquer des erreurs et identifier des vulnérabilités.

avant leur déploiement. Les outils DAST peuvent être intégrés dans les environnements de test et les pipelines CI/CD pour automatiser les analyses de sécurité.

VULNÉRABILITÉS DÉTECTÉES PAR LE DAST ET LEUR RELATION AVEC L'OWASP TOP 10

Rang	Catégorie OWASP Top 10 2021	Description	Relation avec DAST
A01	Bris d'accès (Broken Access Control)	Failles dans les mécanismes de contrôle d'accès, permettant à des attaquants de contourner les autorisations.	**Oui**
A02	Exposition de données sensibles (Sensitive Data Exposure)	Mauvaise gestion de la protection des données sensibles, comme les informations personnelles ou financières.	Potentiel
A03	Injection	Failles liées à l'injection de code non contrôlé dans des requêtes SQL, des scripts ou des systèmes.	**Oui**
A04	Conception peu sûre (Insecure Design)	Problèmes architecturaux liés à une conception de sécurité insuffisante.	Non
A05	Erreurs de configuration de sécurité (Security Misconfiguration)	Défauts dans les configurations qui exposent les systèmes à des attaques.	**Oui**
A06	Utilisation de composants présentant des vulnérabilités connues (Vulnerable and Outdated Components)	Risques liés à l'utilisation de bibliothèques, frameworks ou composants logiciels vulnérables ou obsolètes.	Non
A07	Identification et authentification insuffisantes (Identification and Authentication Failures)	Problèmes dans les mécanismes d'authentification, permettant des accès non autorisés.	Potentiel
A08	Échec de l'intégrité des logiciels et des données (Software and Data Integrity Failures)	Problèmes liés à la falsification de logiciels ou de données pendant les processus de mise à jour ou de déploiement.	Non
A09	Insuffisance des journaux et de la surveillance (Security Logging and Monitoring Failures)	Absence de surveillance et de journalisation suffisante pour détecter et réagir aux incidents de sécurité.	Non

A10	Défaillances du contrôle côté serveur (Server-Side Request Forgery - SSRF)	Failles permettant à un attaquant d'envoyer des requêtes malveillantes à un serveur tiers.	**Oui**

ANALYSE DE LA COMPOSITION LOGICIELLE (SCA)

QU'EST-CE QUE LE SCA?

Le **SCA (Software Composition Analysis)** est une méthode de test qui analyse les composants tiers et les bibliothèques « open source » utilisés dans une application pour identifier les vulnérabilités connues. Le SCA aide à gérer les risques liés à l'utilisation de composants tiers.

LES ÉTAPES DES TESTS DE SÉCURITÉ (SCA)

1. **Identification des composants :** Lister tous les composants tiers utilisés dans l'application.
2. **Scan des composants :** Utiliser des outils SCA pour effectuer le balayage des composants et identifier les vulnérabilités connues.
3. **Évaluation des vulnérabilités :** Examiner les vulnérabilités détectées pour évaluer leur impact.
4. **Mise à jour ou remplacement des composants :** Mettre à jour ou remplacer les composants vulnérables.
5. **Surveillance continue :** Mettre en place une surveillance continue des composants pour détecter les nouvelles vulnérabilités.

FONCTIONNEMENT DES OUTILS SCA

Les outils SCA balayent les composants tiers en les comparant à des bases de données de vulnérabilités connues (Ex. : CVE) pour identifier les risques. Ils fournissent des informations sur les versions vulnérables et recommandent des correctifs ou des mises à jour.

AVANTAGES ET INCONVÉNIENTS DES TESTS SCA

AVANTAGES :

- Identification rapide des vulnérabilités connues dans les composants tiers.
- Aide à gérer les risques liés à l'utilisation de bibliothèques « open source ».
- Permet de rester conforme aux réglementations de sécurité.

INCONVÉNIENTS :

- Ne détecte pas les vulnérabilités non documentées ou « zero-day »[7].
- Peut nécessiter des efforts pour maintenir les composants à jour.

CAS D'UTILISATION COURANTS DU SCA

Le SCA est couramment utilisé pour gérer les risques associés à l'utilisation de composants « open source » dans les applications. Il est souvent intégré dans les pipelines CI/CD pour une surveillance continue.

INTÉGRATION DU SCA DANS LE SDLC

L'intégration du SCA dans le SDLC permet de gérer les risques liés aux composants tiers dès les premières phases du développement, assurant ainsi que l'application reste sécurisée tout au long de son cycle de vie.

VULNÉRABILITÉS DÉTECTÉES PAR LE SCA ET LEUR RELATION AVEC L'OWASP TOP 10

Rang	Catégorie OWASP Top 10 2021	Description	Relation avec SCA
A01	Bris d'accès (Broken Access Control)	Failles dans les mécanismes de contrôle d'accès permettant à des attaquants de contourner les autorisations.	Non

[7] Zero-day : Vulnérabilité inconnue au moment où elle est découverte. Ce type de vulnérabilité est particulièrement dangereuse, car il n'existe pas encore de correctif ou de mise à jour pour la corriger, laissant les systèmes exposés aux attaques jusqu'à ce qu'un correctif soit disponible.

A02	Exposition de données sensibles (Sensitive Data Exposure)	Mauvaise gestion de la protection des données sensibles, comme les informations personnelles ou financières.	**Oui**
A03	Injection	Failles liées à l'injection de code non contrôlé dans des requêtes SQL, des scripts ou des systèmes.	Non
A04	Conception peu sûre (Insecure Design)	Problèmes architecturaux liés à une conception de sécurité insuffisante.	Non
A05	Erreurs de configuration de sécurité (Security Misconfiguration)	Défauts dans les configurations qui exposent les systèmes à des attaques.	Potentiel
A06	Utilisation de composants présentant des vulnérabilités connues (Vulnerable and Outdated Components)	Risques liés à l'utilisation de bibliothèques, frameworks ou composants logiciels vulnérables ou obsolètes.	**Oui**
A07	Identification et authentification insuffisantes (Identification and Authentication Failures)	Problèmes dans les mécanismes d'authentification, permettant des accès non autorisés.	Non
A08	Échec de l'intégrité des logiciels et des données (Software and Data Integrity Failures)	Problèmes liés à la falsification de logiciels ou de données pendant les processus de mise à jour ou de déploiement.	**Oui**
A09	Insuffisance des journaux et de la surveillance (Security Logging and Monitoring Failures)	Absence de surveillance et de journalisation suffisante pour détecter et réagir aux incidents de sécurité.	Non
A10	Défaillances du contrôle côté serveur (Server-Side Request Forgery - SSRF)	Failles permettant à un attaquant d'envoyer des requêtes malveillantes à un serveur tiers.	Non

COMPARAISON ENTRE SAST, DAST ET SCA

Tests de sécurité des applications	SAST	DAST	SCA
Couverture	✓	✓	✓
Faibles faux positifs	*	✓	*
Exploitabilité	✓	✓	
Visibilité du code	✓		
Conseils de remédiation	✓	✓	✓
Intégration au SDLC	✓	✓	✓
Support étendu des plateformes	✓	✓	✓
Axé sur les développeurs	✓	✓	✓
Facile à configurer	✓		✓

DIFFÉRENCES FONDAMENTALES ENTRE SAST, DAST ET SCA

APPROCHE ET MÉTHODOLOGIE :

SAST	• Analyse le code source ou le « bytecode » sans exécuter le programme. • Identifie les vulnérabilités dès les premières phases du développement. • Utilise des techniques d'analyse statique[8] telles que l'analyse de flux de données, l'analyse de contrôle de flux et la vérification des modèles de conception.

[8] Analyse statique : L'analyse statique du code consiste à examiner le code source sans l'exécuter afin de détecter des vulnérabilités potentielles. Elle est utilisée principalement dans les tests SAST.

DAST	• Teste l'application en cours d'exécution, interagissant avec l'application comme le ferait un utilisateur ou un attaquant. • Identifie les vulnérabilités qui se manifestent uniquement en conditions réelles d'exécution. • Utilise des techniques d'analyse dynamique[9] telles que les tests de pénétration et les analyses de comportement.
SCA	• Analyse des composants tiers pour identifier les vulnérabilités connues et gérer les risques liés à leur utilisation.

NATURE DES VULNÉRABILITÉS DÉTECTÉES :

SAST	• Détecte les vulnérabilités liées à la structure du code, telles que les injections SQL, les XSS, les failles de validation d'entrée et les mauvaises pratiques de codage. • Identification des failles dès le début du cycle de développement.
DAST	• Détecte les vulnérabilités liées à l'exécution de l'application, telles que les erreurs de configuration, les problèmes d'authentification et d'autorisation, et les failles dans les interactions de l'application avec les utilisateurs. • Identification des failles présentes dans l'environnement de production.
SCA	• Vulnérabilités dans les composants tiers et les bibliothèques « open source ».

PHASE DE DÉTECTION :

SAST	• Principalement utilisé pendant les phases de développement et de test préliminaires.
DAST	• Principalement utilisé pendant les phases de test final et en production

[9] Analyse dynamique : L'analyse dynamique consiste à tester une application en cours d'exécution pour identifier des failles de sécurité. Elle est utilisée principalement dans les tests DAST.

SCA	• Utilisé tout au long du SDLC pour gérer les risques liés aux composants tiers.

EXIGENCES TECHNIQUES :

SAST	• Nécessite l'accès au code source de l'application. • Peut être intégré directement dans les IDE et les pipelines CI/CD.
DAST	• Nécessite une version déployée de l'application. • Peut être utilisé pour tester les applications sans accès au code source.
SCA	• Nécessite des outils de gestion des composants tiers et un accès aux bases de données de vulnérabilités.

CHOIX ENTRE SAST, DAST ET SCA EN FONCTION DES BESOINS EN MATIÈRE DE SÉCURITÉ

DÉVELOPPEMENT AGILE ET DEVOPS :

SAST	Idéal pour les environnements de développement rapide où les vulnérabilités doivent être détectées et corrigées dès que possible. Intégration dans les pipelines CI/CD pour une détection précoce.
DAST	Complémentaire à SAST pour tester les « builds » déployées et les environnements de production, assurant ainsi que les vulnérabilités introduites en fin de cycle sont également détectées.
SCA	Essentiel dans des environnements où de nombreux composants tiers et bibliothèques « open source » sont utilisés. Le SCA permet de s'assurer que ces composants sont à jour et ne contiennent pas de vulnérabilités connues, ce qui est critique dans un flux DevOps où la réutilisation de code est courante.

TYPES D'APPLICATIONS :

SAST	Recommandé pour les applications complexes avec beaucoup de code personnalisé. Utile pour les applications critiques où la sécurité du code source est primordiale.

DAST	Recommandé pour les applications web et les API où les interactions utilisateurs sont fréquentes et les configurations de production peuvent introduire des vulnérabilités.
SCA	Indispensable pour les applications qui intègrent des bibliothèques « open source » ou des composants tiers. Le SCA est crucial pour les environnements où la sécurité des dépendances logicielles est aussi importante que celle du code propriétaire.

RESSOURCES ET BUDGET :

SAST	Nécessite des développeurs formés à l'analyse statique et à la correction de code. Les outils peuvent varier en coût, mais sont souvent intégrés dans les environnements de développement.
DAST	Peut nécessiter des spécialistes en sécurité pour interpréter les résultats et effectuer des tests de pénétration manuels. Les outils peuvent être coûteux, surtout ceux avec des capacités avancées de tests dynamiques.
SCA	Peut nécessiter un investissement initial pour la mise en place d'outils de gestion des composants tiers, mais il permet de réduire les coûts à long terme en évitant les vulnérabilités dans les bibliothèques « open source », qui sont souvent une source majeure de risques.

UTILISATION COMBINÉE DU SAST, DAST ET SCA POUR UNE APPROCHE COMPLÈTE DE LA SÉCURITÉ DES APPLICATIONS

Pour une stratégie de sécurité complète, l'utilisation combinée du SAST, du DAST et du SCA est fortement recommandée. Cette approche permet de tirer parti des avantages de chaque type de test, couvrant ainsi un large éventail de vulnérabilités. L'intégration de ces trois méthodes tout au long du cycle de vie du développement logiciel (SDLC) garantit une protection optimale des applications contre les menaces de sécurité.

CYCLE DE DÉVELOPPEMENT :

Phase de conception et de développement :

SAST	Intégrer le SAST dès le début du développement pour analyser le code source. Cela permet d'identifier les vulnérabilités structurelles avant qu'elles ne deviennent plus complexes et coûteuses à corriger. Le SAST est particulièrement efficace pour détecter les erreurs de codage, les failles de conception et les mauvaises pratiques de sécurité.
SCA	Simultanément, utiliser le SCA pour scanner les composants tiers et les bibliothèques « open source » dès leur intégration dans le projet. Le SCA permet d'identifier les vulnérabilités connues dans les dépendances logicielles, réduisant ainsi les risques liés à l'utilisation de ces composants.

Phase de test :

SAST et SCA	Pendant les phases intermédiaires du développement, continuer à utiliser le SAST pour analyser les nouvelles modifications de code. Le SCA doit également être utilisé pour s'assurer que les versions des composants tiers sont à jour et ne contiennent pas de vulnérabilité connue.
DAST	Utiliser le DAST pour tester les « builds » déployées dans un environnement de test. Le DAST simule des attaques réelles pour identifier les vulnérabilités qui se manifestent uniquement en conditions réelles d'exécution. Cela complète les tests statiques en offrant une perspective dynamique sur la sécurité de l'application.

Phase de production :

DAST	Une fois l'application déployée en production, le DAST est utilisé pour tester en continu les vulnérabilités potentielles qui pourraient être exploitées en environnement réel. Cela inclut la détection des failles de configuration, des erreurs d'authentification et des vulnérabilités liées aux interactions utilisateur.
SCA	Continuer à surveiller les composants tiers en production avec le SCA. Si de nouvelles vulnérabilités sont découvertes dans les bibliothèques « open source » utilisées, elles doivent être corrigées immédiatement pour prévenir tout risque de sécurité.

AUTOMATISATION ET INTÉGRATION CI/CD :

SAST	Automatiser les numérisations SAST dans les pipelines CI/CD pour effectuer des analyses continues après chaque « commit » ou chaque « build ». Cette automatisation permet de détecter rapidement les vulnérabilités et de les corriger avant qu'elles ne deviennent un problème en production. Les résultats des analyses doivent être intégrés dans les outils de gestion des tickets pour un suivi efficace.
DAST	Automatiser les numérisations DAST après les déploiements dans les environnements de test et de production. Utiliser des scripts et des outils d'orchestration pour déclencher des tests dynamiques, analyser les résultats et s'assurer que toutes les vulnérabilités détectées sont corrigées avant la mise en production.
SCA	Intégrer les numérisations SCA dans le pipeline CI/CD pour surveiller en continu les dépendances logicielles et identifier les nouvelles vulnérabilités dès leur apparition. Cette surveillance continue permet de garantir que les composants tiers utilisés dans l'application restent sécurisés et conformes.

FORMATION ET SENSIBILISATION :

SAST	Former les développeurs aux techniques et outils de SAST pour qu'ils puissent écrire du code sécurisé dès le départ. Cette formation doit inclure la compréhension des vulnérabilités courantes et la manière de les éviter lors du développement.
DAST	Sensibiliser les équipes d'assurance qualité et de sécurité aux techniques et outils de DAST pour qu'elles puissent effectuer des tests dynamiques efficaces. Elles doivent être capables d'interpréter les résultats des tests DAST et de collaborer avec les développeurs pour corriger les vulnérabilités identifiées.
SCA	Former les équipes de développement à l'utilisation des outils SCA pour qu'elles puissent gérer les dépendances logicielles de manière proactive. Elles doivent comprendre l'importance de maintenir les composants tiers à jour et de surveiller les vulnérabilités connues.

GESTION DES VULNÉRABILITÉS :

SAST, DAST et SCA	Mettre en place un processus global de gestion des vulnérabilités qui englobe les vulnérabilités identifiées par le SAST, le DAST et le SCA. Ce processus doit inclure la priorisation des vulnérabilités en fonction de leur criticité, la

	documentation des corrections apportées et la vérification de l'efficacité de ces corrections à travers des réanalyses régulières.
SCA	Assurer un suivi continu des vulnérabilités dans les composants tiers grâce au SCA. Lorsque des vulnérabilités sont découvertes dans des bibliothèques « open source », des plans de correction doivent être immédiatement mis en œuvre, incluant des mises à jour ou le remplacement des composants vulnérables.

MEILLEURES PRATIQUES POUR LES TESTS DE SÉCURITÉ DES APPLICATIONS

DÉFINITION D'UNE STRATÉGIE DE TEST DE SÉCURITÉ DES APPLICATIONS

Élaborer une stratégie claire et structurée pour les tests de sécurité des applications, incluant des objectifs, des méthodes de test et des critères de réussite. Une stratégie bien définie permet de s'assurer que tous les aspects de la sécurité sont couverts et que les tests sont efficaces.

SÉLECTION DES BONS OUTILS SAST, DAST ET SCA

Choisir des outils adaptés aux besoins spécifiques de l'organisation, en tenant compte des types d'applications, des langages de programmation et des exigences de sécurité. Il est important de sélectionner des outils qui s'intègrent bien dans l'environnement de développement et de test existant.

INTÉGRATION DES TESTS DE SÉCURITÉ DANS LE PROCESSUS DE DÉVELOPPEMENT

Intégrer les tests de sécurité tout au long du SDLC, depuis la phase de conception jusqu'à la mise en production et de la maintenance continue. Cette intégration permet de détecter et de corriger les vulnérabilités dès qu'elles apparaissent, réduisant ainsi les risques de sécurité.

FORMATION DES DÉVELOPPEURS AUX PRATIQUES DE SÉCURITÉ DES APPLICATIONS

POURQUOI LA FORMATION EST-ELLE CRUCIALE?

La formation des développeurs aux pratiques de sécurité est essentielle pour plusieurs raisons :

- **Prévention des vulnérabilités :** Les développeurs formés peuvent éviter d'introduire des vulnérabilités dans le code.
- **Détection précoce :** Ils peuvent identifier et corriger les problèmes de sécurité dès les premières phases du développement.
- **Culture de sécurité :** La formation promeut une culture de sécurité au sein de l'équipe, augmentant ainsi la vigilance et la responsabilité collective.

CONTENU DE LA FORMATION

Une formation efficace en sécurité des applications devrait couvrir plusieurs domaines clés :

1. **Principes fondamentaux de la sécurité des applications**
   - Compréhension des concepts de base tels que la confidentialité, l'intégrité et la disponibilité.
   - Familiarisation avec les principales menaces et vulnérabilités (ex. : injections SQL, XXE, CSRF[10], SSRF,[11] etc.).
2. **Meilleures pratiques de codage sécurisé**
   - Techniques de validation et de nettoyage des entrées.
   - Utilisation sécurisée des API et des bibliothèques tierces.
   - Gestion sécurisée des sessions et des authentifications.
3. **Utilisation des outils SAST et DAST**

[10] Cross-Site Request Forgery (CSRF) est une attaque qui force un utilisateur authentifié à exécuter des actions non désirées sur une application web sans qu'il s'en rende compte. En utilisant des techniques d'ingénierie sociale, comme l'envoi de liens malveillants par courriel ou via une discussion, un attaquant peut manipuler l'utilisateur pour effectuer des actions critiques, telles que le transfert de fonds ou la modification de ses informations personnelles. Si la victime est un administrateur, l'attaque peut potentiellement compromettre l'ensemble de l'application web.

[11] SSRF (Server-Side Request Forgery) est une vulnérabilité où un attaquant parvient à manipuler un serveur pour envoyer des requêtes vers des ressources externes ou internes, non prévues par l'application. En exploitant cette faille, l'attaquant peut accéder à des services internes du serveur, récupérer des informations sensibles ou utiliser le serveur pour lancer des attaques sur d'autres systèmes.

- Introduction aux outils SAST et DAST.
- Comment intégrer ces outils dans le cycle de développement.
- Interprétation des résultats des analyses SAST et DAST.

4. **Gestion des vulnérabilités**
   - Processus de correction des vulnérabilités identifiées.
   - Stratégies de gestion des risques associés aux vulnérabilités.
   - Documentation et communication des corrections effectuées.
5. **Normes et cadres de sécurité**
   - Familiarisation avec les normes de sécurité telles que OWASP Top 10, CWE[12]/SANS Top 25.
   - Connaissance des cadres de conformité réglementaire pertinents (ex. : GDPR, PCI-DSS).

MÉTHODES DE FORMATION

1. **Sessions de formation en classe**
   - Ateliers et séminaires dirigés par des experts en sécurité.
   - Cours en ligne interactifs et webinaires.
2. **Formation pratique**
   - Exercices pratiques et laboratoires de sécurité.
   - Simulations d'attaques et de défense.
   - Participation à des programmes de « bug bounty » internes ou externes.
3. **Programmes de certification**
   - Encourager les développeurs à obtenir des certifications en sécurité (ex. : Certified Secure Software Lifecycle Professional [CSSLP], Offensive Security Certified Professional [OSCP]).
4. **Ressources continues**
   - Accès à des bases de connaissances et des ressources en ligne.
   - Participation à des forums et des communautés de sécurité.
   - Mises à jour régulières sur les nouvelles menaces et vulnérabilités.

[12] CWE (Common Weakness Enumeration) : est un répertoire centralisé qui catalogue les types courants de faiblesses dans les logiciels qui peuvent conduire à des vulnérabilités. Elle est utilisée comme référence pour les outils SAST.

GESTION DES VULNÉRABILITÉS IDENTIFIÉES

POURQUOI UNE GESTION EFFICACE DES VULNÉRABILITÉS EST ESSENTIELLE

La gestion des vulnérabilités identifiées est un élément crucial de la stratégie de sécurité des applications. Sans un processus bien défini pour traiter ces vulnérabilités, même les meilleures pratiques de détection (SAST, DAST, SCA) ne suffisent pas à assurer la sécurité des applications. Une gestion efficace permet de :

- **Réduire les risques :** Prioriser les vulnérabilités critiques et les corriger rapidement pour réduire les risques d'exploitation.
- **Maintenir la conformité :** Assurer que les applications respectent les normes de sécurité et les réglementations en vigueur.
- **Préserver la réputation :** Éviter les violations de données et autres incidents de sécurité qui pourraient nuire à la réputation de l'organisation.

ÉTAPES DU PROCESSUS DE GESTION DES VULNÉRABILITÉS

1. **Identification et enregistrement des vulnérabilités**
    - **Centralisation des résultats :** Toutes les vulnérabilités identifiées par les outils SAST, DAST et SCA doivent être centralisées dans un système de gestion des vulnérabilités ou un outil de suivi des tickets. Ce processus permet de consolider les données provenant de différentes sources et de maintenir une vue d'ensemble.
    - **Documentation détaillée :** Chaque vulnérabilité doit être documentée avec des détails sur sa nature, sa source (SAST, DAST, SCA), son emplacement dans le code ou l'application et ses conséquences potentielles.
2. **Évaluation et priorisation des vulnérabilités**
    - **Évaluation du risque :** Chaque vulnérabilité doit être évaluée en fonction de sa gravité, de la facilité avec laquelle elle peut être exploitée et de l'impact potentiel sur l'application. L'utilisation de cadres tels que *Common Vulnerability Scoring System* (CVSS) peut aider à standardiser cette évaluation.
    - **Priorisation :** Les vulnérabilités doivent être classées par ordre de priorité, avec les vulnérabilités critiques traitées en premier. Les facteurs de priorisation peuvent inclure l'impact sur la sécurité, la conformité réglementaire et l'exposition publique.

3. **Correction des vulnérabilités**
    - **Planification des corrections :** Un plan de correction doit être établi pour chaque vulnérabilité, en définissant les responsables, les délais et les étapes de la correction. Les vulnérabilités critiques doivent être corrigées immédiatement, tandis que les vulnérabilités de moindre priorité peuvent être planifiées dans les cycles de développement futurs.
    - **Implémentation des corrections :** Les équipes de développement doivent appliquer les corrections nécessaires, que ce soit en modifiant le code, en mettant à jour les bibliothèques « open source » (dans le cas du SCA) ou en ajustant les configurations de sécurité.
    - **Collaboration interéquipes :** La correction des vulnérabilités peut nécessiter une collaboration entre les équipes de développement, de sécurité, et d'exploitation pour s'assurer que les changements apportés n'introduisent pas de nouvelles vulnérabilités ou n'affectent pas les performances de l'application.
4. **Validation post-correction**
    - **Réanalyse et tests :** Après la correction, il est essentiel de réanalyser l'application en utilisant les outils SAST, DAST et SCA pour vérifier que la vulnérabilité a été correctement corrigée et qu'aucune nouvelle vulnérabilité n'a été introduite.
    - **Tests de régression :** Effectuer des tests de régression pour s'assurer que la correction n'a pas affecté d'autres parties de l'application ou introduit de nouveaux problèmes fonctionnels.
5. **Surveillance continue et réévaluation**
    - **Surveillance continue :** Les vulnérabilités doivent être surveillées en continu, même après correction, pour détecter tout signe d'exploitation ou de réapparition. Cette surveillance est particulièrement importante pour les vulnérabilités liées aux composants tiers (SCA) qui peuvent nécessiter des mises à jour régulières.
    - **Réévaluation périodique :** Réévaluer périodiquement les vulnérabilités non critiques qui n'ont pas encore été corrigées pour s'assurer qu'elles ne deviennent pas plus graves ou exploitables avec le temps.
6. **Communication et documentation**
    - **Rapport de vulnérabilité :** Documenter chaque étape du processus de gestion des vulnérabilités, y compris l'identification, l'évaluation, la correction, et la validation. Cette documentation est essentielle pour les audits de sécurité et la conformité réglementaire.
    - **Communication aux parties prenantes :** Informer les parties prenantes, y compris la direction, les équipes de développement et les clients (si nécessaire), des vulnérabilités critiques identifiées et des actions prises pour les corriger. Une communication transparente est essentielle pour maintenir la confiance et assurer la coordination entre les équipes.

7. **Amélioration continue**
    - **Leçons apprises :** Après la correction des vulnérabilités, il est important de tirer des leçons de chaque incident pour améliorer les pratiques de sécurité et prévenir la récurrence de vulnérabilités similaires.
    - **Mise à jour des processus :** Ajuster et affiner les processus de développement, de gestion des vulnérabilités et de formation des équipes en fonction des leçons apprises et des nouvelles menaces identifiées.

OUTILS SAST, DAST ET SCA POPULAIRES

Outils SAST	• Checkmarx • Fortify • SonarQube • Veracode
Outils DAST	• OWASP Zed Attack Proxy • Burp Suite • Acunetix • AppScan (IBM Security) • Netsparker • Rapid7 AppSpider
Outils SCA	• Black Duck • WhiteSource • Snyk • Sonatype Nexus

CONCLUSION

La sécurité des applications est un aspect critique du développement logiciel moderne. Les tests de sécurité statique (SAST) et dynamique (DAST) jouent des rôles complémentaires dans la détection et la correction des vulnérabilités. En intégrant ces tests tout au long du cycle de vie du développement logiciel (SDLC), les organisations peuvent renforcer la sécurité de leurs applications et protéger les données de leurs utilisateurs.

IMPORTANCE D'UNE APPROCHE CONTINUE DES TESTS DE SÉCURITÉ DES APPLICATIONS

Une approche continue des tests de sécurité permet de maintenir un niveau élevé de sécurité tout au long de la vie d'une application. Les menaces évoluent constamment et les tests de sécurité doivent être réalisés régulièrement pour garantir que les applications restent protégées contre les nouvelles vulnérabilités.

AVANTAGES DE L'ADOPTION D'UNE CULTURE DE SÉCURITÉ DÈS LE DÉBUT DU DÉVELOPPEMENT

L'adoption d'une culture de sécurité dès le début du développement permet de détecter et de corriger les vulnérabilités plus tôt, réduisant ainsi les coûts et les efforts nécessaires pour les corriger. Cela contribue également à sensibiliser les développeurs à l'importance de la sécurité, les incitant à écrire du code plus sûr.

RÉFÉRENCES

https://owasp.org/www-project-top-ten/
https://snyk.io/fr/learn/application-security/static-application-security-testing/
https://www.techmagic.co/blog/dast/
https://medium.com/@ajay.monga73/understanding-sast-dast-and-sca-essential-layers-of-application-security-e4a06d3e7f75
https://docs.sonarsource.com/sonarqube/latest/
https://portswigger.net/burp/documentation
https://www.zaproxy.org/docs/
https://owasp.org/www-project-samm/
https://www.synopsys.com/blogs/software-security/sast-iast-dast-rasp-differences.html
https://docs.sonarqube.org/
https://www.checkmarx.com/resources/
https://www.veracode.com/products/binary-static-analysis-sast
https://www.opentext.com/what-is/sast
https://www.mend.io/blog/sast-static-application-security-testing/
https://www.mend.io/software-supply-chain-security/
https://docs.mend.io/bundle/wsk/page/mend_platform_onboarding_overview.html
https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/test-management/7.0.3?topic=overview
https://docs.gitlab.com/ee/user/application_security/sast/
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf
https://www.veracode.com/security/dast-test
https://www.veracode.com/blog/intro-appsec/sast-vs-dast-security-testing-unveiling-differences
https://owasp.org/www-chapter-dorset/assets/presentations/2020-01/20200120-OWASPDorset-ZAP-DanielW.pdf
https://portswigger.net/burp/documentation
https://www.acunetix.com/vulnerability-scanner/
https://www.netsparker.com/
https://www.rapid7.com/fundamentals/dast/
https://www.qualys.com/apps/web-app-scanning/
https://www.hcl-software.com/appscan
https://www.hcl-software.com/appscan/solutions/dynamic-application-security-testing-dast?referrer=www.hcl-software.com
https://www.hcl-software.com/appscan/solutions/static-application-security-testing-sast?referrer=www.hcl-software.com
https://www.hcl-software.com/appscan/solutions/software-composition-analysis-sca?referrer=www.hcl-software.com
https://docs.gitlab.com/ee/user/application_security/dast/

https://www.invicti.com/blog/web-security/dast-vs-sast-fact-check-on-static-and-dynamic-application-security-testing/
https://snyk.io/fr/product/open-source-security-management/
https://www.mend.io/open-source-security/
https://github.blog/security/supply-chain-security/sca-vs-sast-what-are-they-and-which-one-is-right-for-you/
https://github.blog/enterprise-software/secure-software-development/the-architecture-of-sast-tools-an-explainer-for-developers/
https://github.blog/security/supply-chain-security/secure-your-software-supply-chain-and-protect-against-supply-chain-threats-github-blog/
https://owasp.org/www-project-dependency-check/
https://jfrog.com/xray/
https://www.veracode.com/products/software-composition-analysis
https://www.sonarsource.com/products/sonarqube/

RÉVISIONS

Date	Action	Auteur	Version
2024-09-20	Version initiale	Kamel Chraiti CESI de l'UQ	1.0