< CARTOGRAPHIE DES TECHNOLOGIES >

GROUPE DE TRAVAIL CYBERSECURITE AGILE

CAMPUS CYBER 2025

< SOMMAIRE >

1. INTRODUCTION

Au sein du Campus Cyber, le Groupe de travail Cybersécurité Agile a été constitué pour étudier les meilleures pratiques et les transformations nécessaires à l'atteinte d'un bon niveau de sécurité dans les cycles de développement agiles. Trois axes de travail ont été identifiés qui ont donné lieu à trois chantiers au sein du groupe :

1. **Gouvernance**
2. **Security Champions & SME**
3. **Technologies**

Ce document est le livrable principal du troisième chantier dit « Technologies ».

Les discussions et entretiens réalisés auprès des acteurs de la communauté d'intérêt constituée sur ce sujet ont mis en évidence l'importance de l'aspect technologique dans la transformation dont il est question.
Il est une évidence que certaines activités de sécurité, identifiées notamment dans les travaux du chantier Gouvernance, requièrent la mise en œuvre d'outillages spécifiques. C'est le cas, par exemple et notamment, des activités d'analyse sécurité automatisées dans le cycle de vie des applications telles que le SAST ou le DAST.

De plus, les retours d'expérience collectés ont montré que certains outils ou approches technologiques s'avéraient être des leviers efficaces de la transformation pour fluidifier l'intégration de la sécurité dans les cycles et supporter des approches organisationnelles spécifiques, telles que le modèle « Security Champion » décrit dans les travaux du chantier éponyme.

Ce document synthétise les réflexions et travaux menés par le groupe de travail, il poursuit deux objectifs principaux :

- **Présenter une cartographie des technologies d'intérêt dans le contexte.**

Il est ressorti de nos travaux que le nombre de briques technologiques impliquées dans la sécurité pour les cycles de vie agiles est relativement important, affublées de nombreux acronymes parfois obscurs, et possédant des interactions, adhérences ou recoupements fréquents. Ainsi, dans un premier temps, nous souhaitons apporter une clarification de cet écosystème en identifiant et décrivant chacun de ces composants et leurs connexions. Il s'agit notamment d'illustrer comment ces éléments viennent supporter les approches décrites dans les deux autres chantiers du groupe de travail, processus et organisation.

Finalement, la présentation de cet ensemble de technologies dessine les prémices d'une forme d'architecture de référence qui permettra au lecteur de la décliner dans son contexte.

- **Donner des orientations pour accompagner les choix des organisations.**

De manière quasi-systématique, les questionnements des organisations participant aux travaux sur le sujet ont été relatifs à la feuille de route à adopter, les priorités, les étapes. Face à cela, il s'avère que le plan à adopter reste quand même assez variable d'une organisation à une autre et dépend aussi de sa montée en maturité telle que décrite par le chantier Gouvernance. Il n'y a pas un seul chemin mais certainement des constantes et des choix à faire en fonction des contextes.

Ainsi, nous souhaitons donc apporter des clés pour accompagner le lecteur dans ces choix. Dans cette optique, nous avons synthétisé pour chaque technologie les retours d'expérience, les apports identifiés et les contraintes qui ont été rencontrées sur le terrain par les différents participants.
Pour certains cas, nous éclairons certaines décisions à prendre, comme l'orientation à choisir en analyse SAST ou IAST, pour finalement étudier les critères qui permettront à une organisation de définir ses priorités dans sa montée en maturité technologique.

REMERCIEMENTS

Les rédacteurs de ce livrables, sont (par ordre alphabétique) :

- François-Xavier de Bouët du Portal, EURO-INFORMATION
- Jean-Christophe Delabarre, I-TRACING
- Olivier Dupuy d'Uby, IBM
- Arnaud Lacroix, SANOFI

Ont par ailleurs contribué aux travaux de réflexion ou à la rédaction :

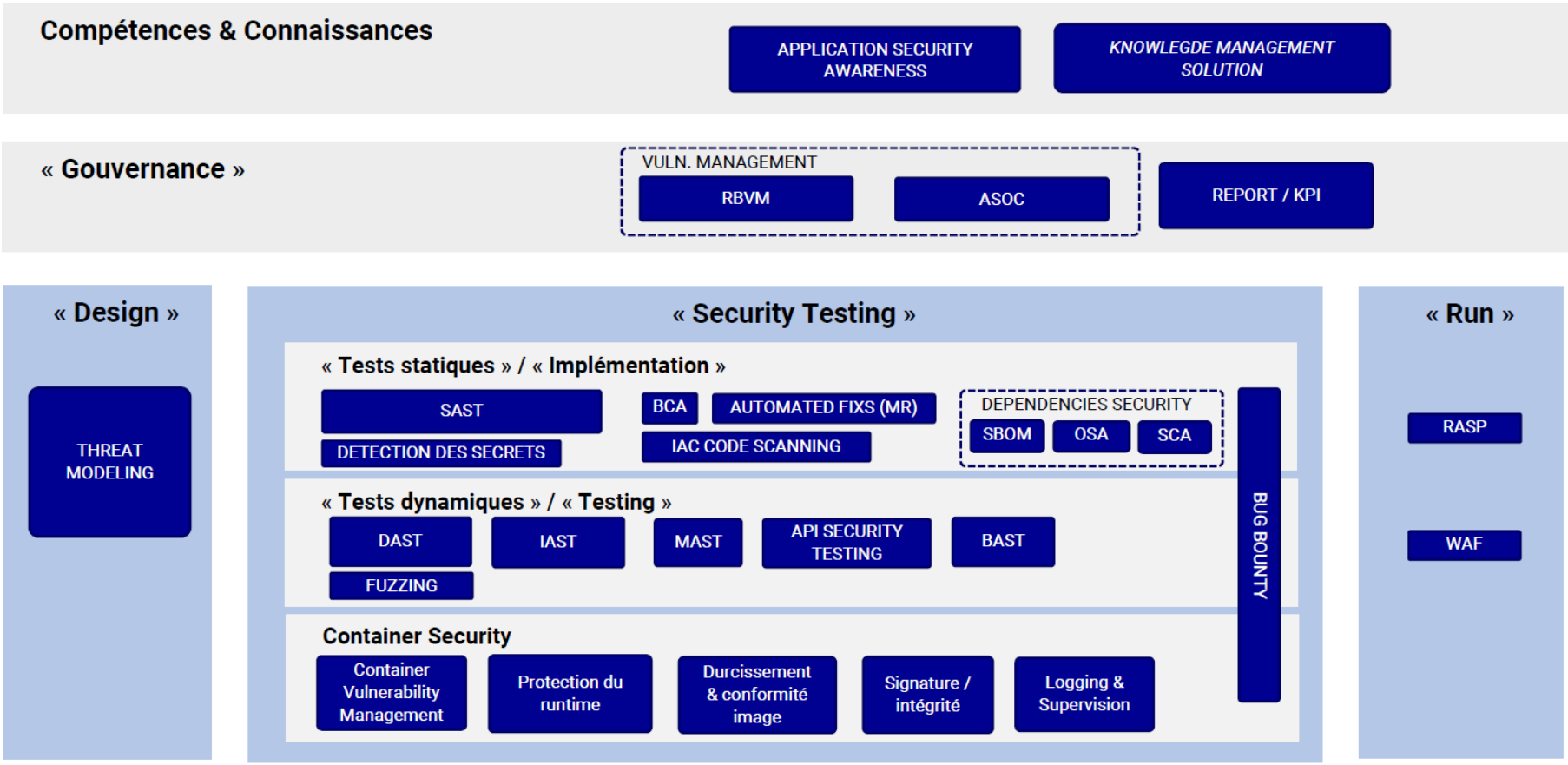
- Thomas Berson, BPCE
- Edith Sandrine Nguessan, ORANGE
- Sophie Thenot, LA FRANCAISE DES JEUX
- Benoit Trinité, RENAULT

2. CARTOGRAPHIE DES TECHNOLOGIES

Les technologies de support à la Cybersécurité Agile prennent différentes approches. Il y a celles :

- Permettant la diffusion de la connaissance
- Permettant d'alimenter le décisionnel
- Purement techniques, servant le design, les tests de sécurité et la protection applicative.

Elles sont regroupées sous ces différentes catégories dans le schéma suivant :



3. DESCRIPTION DES TECHNOLOGIES

Les technologies illustrées dans la cartographie sont décrites ci-après.

Remarque générale : Des exemples d'outils implémentant ladite technologie sont fournis à titre d'illustration. Ces exemples n'ont pas vocation à être exhaustifs, ni à mettre en avant un outil plutôt qu'un autre : il s'agit de permettre au lecteur de mieux s'approprier la définition de la technologie, à travers la fourniture d'un exemple d'outil existant.

3.1 GESTION DE LA CONNAISSANCE

3.1.1 APPLICATION SECURITY AWARENESS (ASA)

Cette technologie permet d'entraîner et de former des développeurs aux bonnes pratiques de développement sécurisé, via différents tutoriels et exercices souvent ludiques.

Ces exercices établissent un lien entre une exploitation et sa remédiation, ce qui permet l'acquisition de réflexes de sécurité pour les développeurs. On notera que parmi les solutions actuelles, certaines permettent également d'adresser les équipes DevOps (Terraform, Docker, ...).

Suivant les solutions éditeurs utilisées, il sera possible de :

- Créer différents parcours d'apprentissage pour différentes équipes
- Concevoir ses propres exercices pour s'adapter à son système d'information
- Mettre en place une intégration dans le LMS (Learning Management System).

De plus, certains éditeurs proposent des solutions de détection des vulnérabilités (SAST, DAST...) qui permettent d'avoir un lien vers des explications et des exercices liés aux types de vulnérabilités détectées.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Checkmarx CodeBashing
- Secure Code Warrior
- SecureFlag
- Avatao
- Immersive Labs

3.1.2 KNOWLEGDE MANAGEMENT SOLUTION (KMS)

Un Knowledge Management System (KMS) est une plateforme numérique conçue pour collecter, organiser et partager les connaissances au sein d'une organisation.

Les objectifs d'un KMS sont multiples, il vise notamment à améliorer la collaboration, la productivité, favoriser l'innovation et faciliter la prise de décision.

Il permet de gérer efficacement l'information et le savoir-faire de l'entreprise, en les rendant accessibles donc de décloisonner l'information, et de préserver les connaissances de l'entreprise.

Bien que le KMS ne soit pas spécifiquement dédié à la sécurité, il peut être très utile pour les security champions.

Voici quelques exemples d'utilisation possible :

- Diffuser les meilleures pratiques en matière de cybersécurité
- Documenter leurs retours d'expérience et les problématiques de sécurité
- Collaborer sur des projets de sécurité transverses
- Centraliser les politiques et les procédures de sécurité
- Gérer une base de connaissances sur les incidents passés et leur résolution
- Faciliter la formation continue en sécurité

Cette base de connaissances permet donc aux security champions d'avoir facilement accès à toutes les informations dont ils ont besoin pour remplir leur rôle.
De plus, elle leur offre un moyen efficace de communiquer des informations importantes à toutes les autres équipes, renforçant ainsi la culture de sécurité dans l'ensemble de l'organisation.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Confluence
- Github
- Sharepoint
- XWiki (Open-Source)
- Zoho Wiki

3.2 GOUVERNANCE

3.2.1 APPLICATION SECURITY ORCHESTRATION AND CORRELATION (ASOC)

Cette technologie consiste principalement en l'orchestration des outils de sécurité, et la centralisation des résultats, ce qui lui permet de fournir différents états des lieux de la sécurité et du traitement des remontées au sein du système d'information.
Le fonctionnement plus détaillé de l'ASOC peut être présenté à l'aide des 5 point clés suivants :

1. Intégration des outils de sécurité (SAST, DAST, IAST, SCA, etc).
2. Centralisation des données, les alertes et résultats de scans sont consolidés dans une plateforme unique.
3. Corrélation et déduplication des résultats des différents outils.
4. Consolidation et automatisation des workflows, des processus de tests, d'analyse et de remédiation.
5. Visibilité et reporting, les équipes DevSecOps disposent d'indicateurs consolidés et de tableaux de bord.

Les principaux objectifs des solutions ASOC portent donc sur l'amélioration de la gestion des vulnérabilités et de l'efficacité des processus de sécurité. D'une manière générale, la force de l'ASOC réside dans sa capacité à offrir une très large couverture d'analyse et à détecter de nombreux types de vulnérabilités.En revanche, la priorisation et la corrélation peuvent être complexes, pour les équipes DevSecOps et les security champions.

Avantages principaux

- Les équipes DevSecOps disposent d'indicateurs consolidés et de tableaux de bord pour piloter efficacement la sécurité applicative et démontrer la conformité aux standards
- Les processus AppSec sont consolidés, les workflows prédéfinis peuvent être facilement suivis, améliorant la communication et la réactivité.

Limitations / point d'attention

- Les processus de déduplication et de corrélation sont complexes et constituent un véritable défi aussi bien pour les solutions ASOC que pour les équipes de DevSecOps. Est-il possible de relier une vulnérabilité DAST à une application microservice ?
- La normalisation des différents niveaux de sévérité/criticité est complexe et nécessitera sûrement une personnalisation de la part des équipes utilisatrices afin d'obtenir une priorisation efficace.
- La disponibilité de plugins vers les solutions de sécurité sources, et surtout leur mise à jour régulière sont des critères de choix importants.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Hackuity
- Ivanti Neurons
- Code Dx
- Konducto

3.2.2 RISK BASED VULNERABILITY MANAGEMENT (RBVM)

Cette technologie permet de prioriser le traitement des vulnérabilités, par le calcul d'un score de risque avec une plus grande granularité que le CVSS de base pour intégrer le contexte à la sévérité de la vulnérabilité remontée.

Pour cela, elle va s'appuyer sur la collecte de données provenant de multiples sources internes (inventaire des assets de l'entreprise, logs d'événements de sécurité, …) et externes (bases de données de vulnérabilités, *threat intelligence feeds*, …) afin de fournir une vue la plus large possible.
Elle peut être directement intégrée à la technologie ASOC.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Hackuity
- Ivanti Neurons
- Qualys VMDR

Les principales complexités de mise en œuvre de cette technologie résident dans le volume de données et la multiplicité de sources. L'effort pour harmoniser les données peut rapidement devenir important.
Le choix de l'outil doit également prendre en compte ses capacités d'intégrer toutes les sources de données identifiées.
Il est également essentiel de définir des règles claires de gestion des priorités de remédiation, au risque d'être rapidement débordé par le volume de vulnérabilités remontées. De plus en plus d'outils proposent l'assistance de l'IA dans cette tâche, ce qui rend l'harmonisation des données encore plus cruciale.

3.2.3 REPORT & KPI SOLUTION (RKS)

Cette technologie permet de visualiser les données schématisées relatives à la sécurité applicative à des fins décisionnelles. Elle permet d'optimiser les stratégies de mise en conformité et de formation par une observation macro des tendances sécuritaires. Elle peut être directement intégrée aux RBVM ou ASOC.

Les indicateurs peuvent concerner aussi bien la mesure du déploiement de la stratégie que la mesure de l'impact de la stratégie.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Power BI
- Hackuity
- Ivanti Neurons
- Metabase (Open-Source)
- Redash (Open-Source)
- Google Data Studio

3.3 DESIGN - THREAT MODELING (TMOD)

La modélisation des menaces (*Threat Modeling*) est un processus d'identification, dénombrement et priorisation des menaces potentielles en fonction des risques et contre mesures associées.

Les technologies supportant ce processus sont principalement des représentations visuelles (diagrammes d'architecture, de flux de données ou de processus) et des questionnaires d'évaluation de risque, permettant la priorisation et le suivi des menaces potentielles à traiter.

Certains outils mettent à disposition des bibliothèques prédéfinies de modèles d'architecture et de composants préremplis avec leurs menaces les plus courantes, et permettant d'intégrer le suivi du développement des contremesures dans les outils de gestion de projet.

Les solutions de *Threat Modeling* s'intègrent généralement avec d'autres technologies (ex. formation) et peuvent être mappées avec des tests en aval. Elles servent à concevoir l'application de manière sécurisée en identifiant les menaces.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- IriusRisk
- ThreatModeler
- MS Threat modeling

3.4 SECURITY TESTING – ANALYSES DU CODE SOURCE ET COMPILÉ

Cette section regroupe l'ensemble des technologies d'analyse de code, c'est-à-dire du code applicatif sous forme de fichiers sources ou compilés.

3.4.1 STATIC APPLICATION SECURITY TESTING (SAST)

Le test statique de sécurité des applications (***Static Application Security Testing - SAST***) consiste en l'analyse du code source d'une application avant sa compilation, pour identifier les failles de développement courantes qui pourraient être exploitées par des acteurs malveillants.

La technologie SAST existe depuis plusieurs dizaines d'années. Elle repose sur l'analyse du code source dans plusieurs dimensions (lexicale, syntaxique, sémantique, flux de données, contrôles et configuration), et la recherche de problèmes connus.

De manière générale, la force du SAST réside dans sa capacité à avoir une très large couverture d'analyse et à détecter de nombreux types de vulnérabilités (à conditions évidemment de disposer des jeux de règles adaptés). En revanche, le SAST peut avoir une propension à produire de très nombreux résultats, en particulier des faux-positifs, ce qui peut avoir un aspect rébarbatif pour les équipes de développement.

Les principaux critères de mesure de la technologie SAST sont les langages supportés, le taux de faux positifs et de bruit :

- Les faux positifs sont des résultats remontés par l'outil comme défauts, mais qui n'en sont pas réellement. Ils proviennent principalement de l'absence de prise en compte du contexte d'exécution (par exemple une alerte sur l'absence de sécurisation de cookies, alors que cette partie est prise en charge par le serveur applicatif), ou encore une limitation du moteur d'analyse (par exemple, la non-reconnaissance d'une fonction d'assainissement d'une entrée utilisateur).
- Le bruit comprend les résultats qui sont avérés, mais qui ne présentent pas d'intérêt direct pour la sécurité ou la qualité de l'application.

Avantages principaux :

- *Shift Left* (application de mesure de sécurité le plus en amont du cycle de développement) : l'analyse SAST peut être effectuée dès qu'une ligne de code est écrite. Certaines solutions proposent une analyse temps réel à la façon d'un correcteur orthographique intégré dans l'IDE.
- Le SAST couvre l'ensemble du code, sans besoin de définir un scénario de test au préalable.
- Bien intégré dans la CI, et doté de processus optimisés, le SAST permet une gestion très fine de la sécurité de l'application. La priorisation des défauts est une étape indispensable qui doit être effectuée au fur et à mesure.
- Il est possible de gérer de la même façon les défauts de sécurité et les défauts de qualité.
- Il est possible de personnaliser les règles (à noter que seules certaines solutions permettent cette personnalisation).

Limitations :

- Le temps d'analyse est proportionnel à la taille de l'application testée et aux types de défauts recherchés.
- Le volume de défauts peut être rébarbatif et engendrer une réaction de rejet auprès des équipes de développement.
- Certaines solutions demandent un effort de customisation important, pour s'adapter aux pratiques de développement suivies par les équipes de développement.
- Parce qu'elle prend en compte l'ensemble du code source, l'analyse peut remonter des vulnérabilités concernant des parties de code non-utilisées par l'application en production.
- L'exhaustivité de la couverture est dépendante de l'analyse initiale faite par le fournisseur des règles dans chacun des langages disponibles. Il est difficile pour une solution SAST d'être exhaustif sur de nombreuses technologies à la fois.

Avant de choisir un outil SAST, il est important de vérifier son niveau de compatibilité avec :

- Les langages et les frameworks de programmation utilisés,
- Les référentiels de conformité utilisés (par exemple OWASP Top 10, PCI DSS, ASD STIG).

En plus des contraintes techniques, il faut se baser sur les contraintes organisationnelles, telles que les ressources et les stratégies de traitement des vulnérabilités afin de s'orienter parmi 3 familles d'outils et d'implémentations, listées ci-dessous avec leurs caractéristiques principales :

Outils open-source

- Fréquence de mise à jour, et soutien de la communauté (dépendant de la vitalité de la communauté de développement),
- Transparence du code mis en œuvre,
- Liberté de customisation complète (coûts humains élevés en fonction des changements apportés),
- Responsabilité totale de l'hébergement, les opérations, et la gestion du cycle de vie de la solution.
- Pas de coût de licence, mais selon le type de licence associée à l'outil devoir de publier les adaptations apportées.

Outils du marché on premise

- Accompagnement et support de l'éditeur,
- Possibilité de customisation dépendant de l'outil,
- Coûts de support et de licence,
- Différentes options d'hébergement avec un coût potentiellement élevé,
- Contraintes sur le déploiement des mises à jour.

Outils SaaS

- Accompagnement et support de l'éditeur
- Customisation plus réduite
- Coûts de support et de licence
- Responsabilité d'hébergement déléguée à l'éditeur
- Suivi obligatoire des montées de versions de l'éditeur
- Transfert de données sensibles (code source et informations de sécurité) vers un tiers.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Checkmarx One/SAST
- OpenText Fortify
- Sypnosys Coverity
- Veracode
- Semgrep
- SonarQube
- Snyk

3.4.2 INTERACTIVE APPLICATION SECURITY TESTING (IAST)

L'analyse interactive de sécurité des application (*Interactive Application Security Testing - IAST*) consiste à surveiller le fonctionnement interne d'une application web pendant son exécution afin de détecter des vulnérabilités. L'analyse IAST exploite les capacités d'instrumentation fournies par les moteurs d'exécution tels que. NET *framework* ou la Java Virtual Machine. Dans la majorité des cas, un agent d'analyse est déployé sur l'environnement d'exécution. Lors de l'utilisation légitime ou normale de l'application, l'outil identifie le code source spécifiquement exploité et procède à une analyse sur cette portion pour identifier de potentielles vulnérabilités.

Les techniques de détection mises en œuvre s'apparentent à celles du SAST, mais appliquées uniquement sur les portions de code contribuant à l'exécution. La qualité de l'analyse est grandement améliorée par la connaissance de l'environnement d'exécution et des flux de données dans l'application. L'IAST voit les données qui transitent dans et à travers l'application. C'est un avantage sur les approches SAST (qui ne considèrent jamais les données) et DAST (qui ne considèrent que les entrées et sorties, l'application étant alors une boîte noire).

Par conception, l'analyse IAST ne peut être exécutée que lorsque l'application est en cours d'exécution et donne des résultats quand celle-ci est effectivement utilisée. Ainsi, en règle générale, l'analyse est réalisée pendant les tests fonctionnels sur les environnements de pré-production. L'étendue des résultats de sécurité est directement liée au niveau de couverture des tests fonctionnels, les portions de code qui ne seraient pas activées pendant les tests n'entreront pas dans l'analyse.

L'analyse IAST présente des avantages forts, notamment comparée à l'analyse SAST :

- En premier lieu, il faut noter que les taux de faux-positifs sont drastiquement réduits. Une analyse statique présuppose de multiples schémas d'exécution de l'application et déduit des potentielles vulnérabilités qui n'ont peut-être aucune réalité opérationnelle. Par l'analyse ciblée sur des schémas d'exécution constatés, l'analyse IAST évite ces faux-positifs.
- L'analyse IAST est aussi plus performante et particulièrement adaptée aux langages dynamiques puisqu'elle analyse alors le code dynamiquement généré au moment de l'exécution.
- L'analyse IAST donne une meilleure visibilité des interactions entre composants applicatifs, particulièrement et notamment dans des architectures micro-services. L'analyse identifie les échanges entre les différents composants et représente une vue des interactions. L'analyse pourra également indiquer des appels vers des services externes.
- Enfin, l'analyse IAST dispose d'une visibilité sur les données d'exécution. Il est donc possible de suivre les transferts de données sensibles entre les différents composants de l'application et des vulnérabilités spécifiques telles que leur stockage de manière non chiffrée. Certaines solutions permettent d'injecter des données pour simuler le fonctionnement et les vulnérabilités de l'application.

En revanche, certaines limitations et inconvénients sont communément identifiés :

- L'analyse IAST est réservée aux applications de type web et doit supporter la technologie de l'application : JVM, Python, Node js, .NET… Certaines technologies ne disposant pas de capacités suffisantes d'instrumentation ne pourront pas être analysées. Les solutions du marché diffèrent sur leur couverture des différentes stacks technologiques.
- Les tests fonctionnels interviennent tardivement dans le cycle de développement. Les applications testées doivent être exécutées.
- L'analyse est limitée aux portions de l'application qui sont effectivement activées. La couverture sécurité est donc immédiatement liée à celle des tests fonctionnels. A noter que certains outils IAST remontent des informations sur la couverture de test et indiquent les parties de l'applications qui n'ont pas été stimulées par le test fonctionnel, et qui de fait, n'ont pas été testées au sens de la sécurité.
- L'analyse peut impacter les performances de l'application et reste donc réservée à des environnements hors production. Elle doit aussi être désactivée dans le cas particulier d'évaluation des performances des applications.

Il faut bien différencier les approches IAST et RASP qui peuvent être confondues car toutes deux s'appuient sur les capacités d'instrumentation des applications. L'IAST analyse le code activé pendant un cas d'utilisation légitime de l'application et détecte des vulnérabilités sur le code. Le RASP identifie un comportement déviant de l'application pendant une attaque et détecte la compromission. L'IAST s'appuie de manière intensive sur l'instrumentation et a une proportion bien supérieure à impacter les performances de l'application, le RASP minimise au maximum son impact sur les performances ciblant les entrées/sorties.

En conséquence, l'IAST est réservée à une phase de tests avant mise en production, le RASP est une approche déployée en production qui n'a pas de sens à être utilisée pendant les tests légitimes.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Contrast Security
- SeekerIAST
- Synopsys

IAST ou SAST ?
In fine, les deux types de détection, IAST et SAST, détectent des vulnérabilités dans le code des applications : elles peuvent paraître redondantes. L'analyse SAST est plus fréquemment mise en place dans les organisations, nous constatons au contraire que les outils IAST restent moins implantés. Cela est peut-être lié à l'histoire avec une apparition plus récente de l'IAST. En tout état de cause, une question se pose : vaut-il mieux implémenter un outil SAST, IAST ou les deux ?

En tout premier lieu, il faut tenir compte de la technologie des applications à tester, et vérifier qu'une approche IAST est effectivement compatible. Dans le cas contraire (applications non « web » et/ou pile technologique non compatible), la question ne se pose pas : le SAST reste la seule approche possible pour identifier des vulnérabilités dans le code avec un outil automatisé.

Dans les cas où l'IAST est effectivement envisageable, ce type d'analyse semble présenter des avantages très forts par rapport au SAST, tout particulièrement dans des organisations faiblement sensibilisées à la sécurité et/ou agiles. En effet, l'IAST présente beaucoup moins de faux-positifs, et, de plus, certaines solutions permettent d'illustrer les vulnérabilités identifiées en démontrant l'exploitabilité avec un cas de test généré. Il s'en suit une bien meilleure adoption de l'outil par les équipes. Ce qui diffère du SAST où au contraire, dans ces contextes moins sensibilisés, les taux de faux-positifs ont tendance à provoquer une forme de rejet. Une analyse SAST apparait plus efficace dans des organisations où la sécurité est finement suivie et où les équipes traitent systématiquement les résultats au fil de l'eau.

En revanche, l'analyse IAST est plus tardive dans le cycle de vie. Une analyse SAST peut être lancée beaucoup plus tôt et répond mieux aux impératifs de « shift-left ».

En synthèse : en première intention, si compatible, une analyse IAST apparaît plus avantageuse bien que plus tardive, en particulier dans les équipes peu sensibilisées ou très contraintes par des enjeux de rapidité de livraison.
Ensuite, pour les contextes très sensibles avec des équipes assidues en matière de sécurité, une analyse SAST complémentaire permettra de renforcer la couverture des risques et de mieux anticiper les éventuels défauts en amont du cycle de vie.

Dans tous les cas, le SAST reste incontournable pour les applications non compatibles avec l'IAST.

3.4.3 SECRET DETECTION

La détection des secrets (*Secret Detection*) est une analyse du code source (incluant fichiers de configuration ou même documentation) se focalisant sur la recherche d'identifiants, mots de passe, chaines de connexions, clés d'API, jetons d'authentification, … qui seraient présents en clair dans le code, y compris dans son historique (notamment l'historique des *commits*, voire des commentaires associés).

Outre le contrôle du respect des bonnes pratiques de développement, l'enjeu est de se prémunir d'une diffusion accidentelle ou malveillante d'identifiants sensibles (via une erreur humaine, la compromission d'un poste de développeur, ou de la chaine CI/CD).

Certains outils SAST (et IAC pour le code d'infrastructure) intègrent des règles simples de détection de secrets, comme les mots de passes codés « en dur » ou les entropies élevées de chaines de caractères, mais avec un taux de faux positifs élevé (par exemple avec les images encodées en base64).

Les outils spécialisés en détection de secrets proposent des moteurs de détection plus fins, permettant de prendre en compte des patterns et de catégoriser les secrets détectés (clé d'API, jeton GitHub, chaine de connexion à une base de données,…).

Certains outils proposent également de vérifier la validité et l'exploitabilité des secrets détectés, permettant ainsi d'affiner le niveau de risque et la priorité à donner à la remédiation. Ils permettent aussi d'ajouter ses propres règles de détection.

Bien intégrés à la chaine CI/CD, ces outils permettent d'éviter la propagation malencontreuse de secrets dans la chaîne de développement.

La détection des secrets peut être appliquée aux images de conteneurs, qui peuvent elles aussi renfermer des secrets stockés en clair (par exemple des clés d'API stockées en variables d'environnement, ou dans des fichiers sensibles, oubliés lors de la publication de l'image).

Il est également possible de déporter une partie de ces contrôles au niveau des postes de développeurs afin de bloquer toute soumission de code portant un secret (en intégrant par exemple l'exécution de la détection de secrets dans un *pre-commit hook* dans l'environnement Git).

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Gitguardian
- Gitleaks
- Trufflehog
- Detect-secrets

3.4.4 AUTOMATED FIXES

Certains outils d'analyse et certaines plateformes de gestion de code source fournissent des suggestions de corrections automatiques, en particulier des montées de version de composant open-source (devenu obsolète ou vulnérable).

Ces suggestions prennent la forme d'une nouvelle branche crée automatiquement dans le code source et de *pull request* associée, pour valider le changement proposé.

Cette technologie peut être un levier à la fois simple et puissant pour assurer une gestion régulière du cycle de vie de tous les composants open-source utilisés dans un projet : associée à la chaîne CI/CD sous la forme d'un workflow récurrent (par exemple hebdomadaire), elle permet de proposer automatiquement les montées de versions à l'équipe de développement.

Sur des projets de grande taille avec des centaines de références, cette technologie peut générer beaucoup de notifications. Dans ce cas, il est nécessaire de définir au préalable les bons filtres, les règles de prise en compte et la fréquence de mise à jour avec l'équipe de développement, afin que chacun tire le meilleur parti de ces notifications.

Certains outils de SCA embarquent également cette technologie.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Renovate Bot (dette technique lié à l'obsolescence),
- GitHub Dependabot

Depuis 2024, GitHub a également ajouté des capacités d'IA (Copilot Autofix), permettant de suggérer des corrections d'alertes SAST.
Quelle que soit la technologie, ces suggestions doivent être vérifiées par les développeurs eux-mêmes, qui restent les mieux placés pour valider leur pertinence.

3.4.5 BYTE CODE ANALYSIS (BCA)

L'analyse statique de code compilé *(Byte Code Analysis - BCA*) reprend les principes d'une analyse SAST, mais en se basant sur le code issu de la compilation de langages interprétés (par exemple Java, C#, Python).

Ce type d'analyse ne requiert pas le code source et propose des résultats au plus proche du code qui sera exécuté, en se basant notamment sur le cheminement des données dans l'applications.

Son principal inconvénient est une visibilité moindre sur les remédiations possibles dans le code source.

Elle est applicable notamment aux technologies embarquées.

Certains éditeurs précisent qu'une analyse de *Byte Code* est assez coûteuse en ressources et peut ne pas être adaptée à une intégration dans la chaîne CI/CD.

Sur les gros projets, on la réservera donc plutôt à des analyses nocturnes pour revue manuelle des résultats le lendemain.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Veracode
- HCL AppScan

3.4.6 INFRASTRUCTURE AS CODE SCANNING (IAC SECURITY TESTING)

La technologie *IaC Security Testing* consiste à analyser le code source des configurations d'infrastructure (ex. Ansible, Terraform, Azure Ressource Manager) dans le but d'identifier des défauts de configuration.

Certains outils permettent également de mesurer la conformité aux standards du marché ou aux politiques spécifiques de l'entreprise.

Cette technologie est souvent associée aux outils de CSPM (Cloud Security Posture Management) car elle intervient en amont de la phase de déploiement.

Reporter ou synchroniser les politiques définies sur le cloud vers l'outil de scan IAC permet de rendre les alertes de sécurité IAC pertinentes.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Checkmarx IaC Security
- Snyk IAC
- Prisma Cloud Application Security

3.4.7 SECURITE DES DEPENDANCES ET ANALYSE DE COMPOSITION

La connaissance des composants ou bibliothèques utilisés par une application est essentielle pour assurer la sécurité de l'application elle-même et de son développement.

On distingue 3 technologies :

- Open-Source Analysis – OSA
- Software Bill of Materials - SBOM
- Software Composition analysis - SCA

Les technologies OSA, SBOM et SCA partagent plusieurs objectifs et fonctionnalités clés :

- Identification des composants : elles permettent chacune de recenser les bibliothèques et composants utilisés dans une application, offrant ainsi une vue d'ensemble complète des dépendances.
- Détection des vulnérabilités : elles visent à identifier les vulnérabilités connues dans les composants utilisés, contribuant ainsi à la sécurité globale de l'application.
- Inventaires des licences: elles fournissent des informations sur les licences des composants, ce qui est crucial pour garantir la conformité légale et éviter les risques liés à l'utilisation de logiciels open-source.

Spécificités de chaque technologie :

L'Open-Source Analysis (OSA) se concentre sur l'analyse des paquets open-source ou des images de conteneurs dans des dépôts privés. Cette analyse indépendante permet de rechercher des vulnérabilités déjà rendues publiques et de fournir des détails sur les conditions d'utilisation des paquets. Ces informations sont particulièrement utiles pour les organisations commerciales, car certains composants open-source peuvent interdire un usage commercial ou obliger à la publication de toute modification du code source.

La *Software Bill of Materials* (SBOM) crée un inventaire complet des bibliothèques et logiciels, ainsi que de leurs dépendances. Elle permet d'identifier rapidement si une application est concernée par une vulnérabilité connue d'un composant donné. La fourniture d'une SBOM est requise par certaines législations. Les formats standardisés tels que SPDX, CycloneDX et SWID sont utilisés pour structurer ces informations de manière cohérente et détaillée.

A noter que la fourniture systématique d'une SBOM :

- Fait partie de la proposition de loi européenne sur la cyber-résilience Cyber Resilience Act | Shaping Europe's digital future (europa.eu)
- Est mentionnée dans le Executive Order on Improving the Nation's Cybersecurity | The White House de mai 2021 publié par l'administration des Etats-Unis.

Il existe à ce jour 3 formats standard de SBOM :

- *Software Package Data Exchange* (SPDX®), ISO/IEC 5962:2021, format open-source, JSON, développé par la Linux Foundation. Ce format très granulaire permet d'inclure des annotations au niveau de chaque référence de fichier, et ainsi d'ajouter des éléments de licence, copyright, et origine.
- CycloneDX (CDX), format open-source, JSON et XML, développé par l'*OWASP community*, plus orienté sécurité et plus léger que SPDX.
- Software Identification (SWID), ISO/EC 19770-2 utilisé par plusieurs éditeurs de logiciels, uniquement réservé à des usages d'inventaire car ne contient pas d'information de sécurité ni de licence.

La *Software Composition Analysis* (SCA) consiste à établir une liste des composants tiers, incluant les dépendances directes et transitives. Elle remonte les informations sur les failles connues (CVE) des composants et peut alimenter une SBOM. Certains outils SCA intègrent également les résultats d'une analyse SAST pour vérifier si une bibliothèque vulnérable est effectivement utilisée dans le code source, offrant ainsi une analyse plus fine et pertinente.

Certains outils embarquent les 3 technologies : SBOM, OSA et SCA.

Les principaux critères de choix de ces outils sont :

- La couverture des différents dépôts open-source (composer, Conda, maven, npm, NuGet, PyPI, CRAN, …) en termes de connaissance des vulnérabilités et d'interprétation de dépendances.
- La capacité de reconnaître des « *code snippets* » appartenant à des composants connus.
- La fréquence de mise à jour de la base de vulnérabilités (de l'éditeur ou de la communauté open-source).
- Les informations d'exploitations en cours.
- La suggestion du chemin de remédiation le plus pertinent, en fonction de la criticité des alertes et du réseau de dépendances.

La définition de la priorité de remédiation des vulnérabilités liées aux dépendances peut s'effectuer suivant plusieurs approches :

- Approche simple : toutes les vulnérabilités détectées doivent être corrigées… selon les projets, cette activité peut être remise en cause au regard des coûts engendrés vs les risques évités.
- Approche liée à l'usage : une première analyse de chaque vulnérabilité permet de se concentrer sur les plus critiques, en fonction de l'usage de l'application, et de traiter les autres par la suite. Cette approche permet un premier niveau de priorisation.
- Approche liée au risque effectif : seules les vulnérabilités affectant les fonctionnalités utilisées par l'application doivent être corrigées. Si cette approche permet de réduire le nombre de corrections à appliquer, en se focalisant sur les plus pertinentes. Elle nécessite une connaissance fine des fonctionnalités utilisées par l'application (i.e. du code source sous-jacent). De plus, chaque modification du code étant susceptible de modifier les fonctionnalités utilisées, cette approche nécessite une surveillance continue.

Dans le cadre de cette dernière approche, une analyse SAST peut venir compléter l'analyse SCA en détectant les chemins d'exploitabilité d'un composant vulnérable.

De même une analyse IAST peut permettre d'offrir une vue dynamique des vulnérabilités en lien avec le code en cours d'exécution.

Risques liés aux dépendances utilisées dans les phases de développement et de compilation (build) :
Les dépendances utilisées dans le développement ou la compilation d'un projet sont elles aussi à surveiller, même si elles ne font pas partie de la composition de l'application finale.

En effet, elles sont particulièrement exposées aux attaques de la chaîne d'approvisionnement logiciel (*Supply Chain*) : un composant malveillant téléchargé sur un environnement de développement peut compromettre l'ensemble du projet. Voici quelques risques spécifiques :

- Injection de code malveillant : les dépendances compromises peuvent contenir du code malveillant qui s'exécute lors de la compilation ou du développement, ouvrant des portes dérobées ou exfiltrant des données sensibles.
- Compromission de l'intégrité du code : altération du code source ou des binaires générés, compromettant ainsi l'intégrité et la sécurité de l'application finale.
- Accès non autorisé : les dépendances malveillantes peuvent permettre à des attaquants d'obtenir un accès non autorisé aux systèmes de développement, de compilation ou de déploiement, facilitant des attaques plus larges.

Problématique des dépendances propriétaires :
Pour les cas des développements reposant sur des dépendances propriétaires (i.e. des composants logiciels développés en interne), ces dernières ne seront pas présentes dans les bases de vulnérabilités connues.

Pour pallier cet angle mort, il est essentiel de maintenir un catalogue interne de ces dépendances, et de réaliser l'évaluation de la sécurité de chacune.

3.5 SECURITY TESTING – TESTS DYNAMIQUES

3.5.1 DAST: DYNAMIC APPLICATION SECURITY TESTING

Les tests dynamiques de sécurité des applications (DAST : Dynamic Application Security Testing), permettent d'examiner une application web en cours d'exécution afin de détecter de potentielles vulnérabilités (ex : entrées et sorties non validées, faiblesses de configuration, mauvais traitements des erreurs …)

Le scan DAST des applications est effectué d'un point de vue extérieur (comme un attaquant), les tests sont conçus pour fournir à l'application ciblée des données malveillantes, ou parfois seulement erronées (fuzzing) et en déduire des vulnérabilités à partir de la réponse obtenue.

Généralement, les tests DAST ont lieu sur un environnement de test et non en production afin de faciliter la mise en place de crédentials (surtout sur un environnement critique) et pour éviter de mettre à mal la production tant en termes de performances que de données enregistrées.

L'expérience montre que l'exécution des tests DAST peut nécessiter, pour une application de taille classique, plusieurs heures pour être complète. Voire plus si une application mène à une autre, ce qui entraine un débordement de l'analyse initiale sur un autre périmètre. Cette durée d'analyse n'en fait pas un bon candidat pour une intégration comme un élément bloquant dans un pipeline.

Il est recommandé de prendre du temps afin de paramétrer l'outil pour exclure certains paramètres du scope de l'analyse, ce qui diminue les faux positifs (ex : ne pas altérer l'identifiant de session, ce qui créé une exception liée à l'invalidité de celui-ci).

Le DAST peut nécessiter plus d'accompagnement pour la compréhension des rapports d'analyse du fait de son aspect *blackbox* (la vulnérabilité est remontée sur une requête et pas sur un pan de code). Certains mécanismes d'amélioration des performances peuvent également ne pas faciliter l'analyse, comme l'altération de plusieurs paramètres simultanément, en cas de remontée sur cette requête, il faut éplucher chacun des paramètres altérés pour identifier lequel est en cause.

Dans une démarche *shift left*, il est possible d'utiliser cet outil en mode proxy. Ainsi, lors de la phase de test en local sur la machine de développement, le DAST va tenter d'injecter chacun des champs lors de la navigation et remonte les vulnérabilités identifiées au fur et à mesure.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- OWASP Zap
- Netsparker
- InsightAppSec de Rapid7
- Veracode
- Qualys WAS

3.5.2 FUZZING

Les tests dynamiques d'injection de données (fuzzing) sont des tests qui injectent des données aléatoires dans les entrées d'un programme en fonctionnement. Cela permet de savoir si le programme a des comportements non voulus, et de corriger les défauts si besoin. À noter que ce type de test n'est pas limité aux applications, mais est aussi utilisé pour tester la sécurité des protocoles, des services et des périphériques embarqués. Leur principal atout réside dans la « facilité » de génération de nombreux tests.

Les solutions de fuzzing modernes peuvent analyser la structure du code qu'ils sont censés tester. Ils peuvent générer des milliers de cas de test automatisés par seconde, et marquer chaque chemin que les données prennent à travers le programme. De cette façon, le « fuzzer » obtient une analyse détaillée sur la couverture du code et les entrées qu'il atteint pendant l'exécution de l'application.

Certains outils de la famille scanner web peuvent être appelés « fuzzers » dans certains contextes.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- OWASP ZAP
- American fuzzy lop
- Defensics Fuzz Testing Synopsys

3.5.3 MOBILE APPLICATION SECURITY TESTING (MAST)

Les tests de sécurité des applications mobiles (MAST : Mobile Application Security Testing) sont une combinaison des différentes familles de scan (SAST, DAST, IAST) permettant d'évaluer la sécurité des applications Android ou iOS. Suivant la solution choisie, les types d'analyses peuvent varier. Les applications Android et iOS seront exécutées en condition réelle lors des scans afin d'analyser efficacement les vulnérabilités et la résilience aux attaques.

- Les autorisations,
- Les communications avec l'extérieur,
- Les fonctionnalités potentiellement dangereuses,
- La collusion des applications,
- L'obfuscation,
- La consommation d'énergie excessive,
- Les vulnérabilités logicielles traditionnelles,
- Les communications internes (débogage, GPS, NFC, urls),
- ...

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Mobile Security Framework (MobSF)
- eShard esChecker

3.5.4 API SECURITY TESTING (API ST)

Les tests de sécurité des API (API : interface de programmation d'application) sont une combinaison des différentes familles de scan (SAST, DAST, IAST) permettant d'évaluer la sécurité des API.

Ces tests, outre la détection de vulnérabilités, permettent aussi de s'assurer que les API respectent le contrat d'API, la politique organisationnelle et les bonnes pratiques. Un volet découverte (discovery) peut aussi être fourni par certaines solutions, afin d'identifier le périmètre exposé.

Certaines solutions offrent aussi la possibilité de configurer des tests fonctionnels de sécurité des API (par exemple, l'authentification ou le contrôle d'accès), dans une optique de non régression.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- OWASP ZAP
- Api Audit 42crunch
- API Security Testing Cequence

3.5.5 BEHAVIORAL APPLICATION SECURITY TESTING (BAST)

Les approches BAST mettent en œuvre des tests fonctionnels et métier dans une orientation sécurité. Elles se focalisent sur le test des fonctions pouvant avoir un impact sur la sécurité dans le parcours utilisateur (fonction manipulant des données sensibles, contrôle d'accès, implémentation d'un seuil…). Elles s'appuyent sur des outils pouvant manipuler des interfaces homme / machine.

La technologie BAST requiert que, à l'instar de DAST et IAST, l'application s'exécute. Elle sera donc employée dans les dernières étapes du S-SDLC.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Selenium

3.5.6 CONTAINER SECURITY (CS)

La sécurité des conteneurs vise à protéger les applications conteneurisées ainsi que l'infrastructure sous-jacente contre les vulnérabilités et les attaques potentielles. Cela implique la mise en place de mesures de sécurité et de bonnes pratiques tout au long du cycle de vie de la création au déploiement, jusqu'à l'exécution.

La sécurité des conteneurs doit aussi inclure l'activité de SECRET DETECTION décrite dans une section précédente.

a) Analyse et évaluation des vulnérabilités

Il s'agit du processus d'identification et d'analyse des vulnérabilités publiques de sécurité affectant des composants embarqués dans les images.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Trivy
- Clair,
- Grype

A noter : l'analyse des containers peut se faire soit avant déploiement avec possibilité de bloquer les images ne respectant pas des critère d'acceptance, soit directement sur l'infrastructure de runtime.

b) Protection du runtime

Mise en place de mesures de sécurité pour protéger le runtime des conteneurs contre les intrusions et les attaques.

Ce type de solution peut faire de multiples analyses, comme :

- Analyse des dépendances et identifications des vulnérabilités connus
- Analyse du comportement des containers en cas de dérive, mutation, exécution de processus suspects ou activité réseau suspecte (reverse shell, exfiltration, cryptominer, …).

Il est généralement possible de générer des alertes mais aussi de bloquer les containers en cas de détection d'activités suspectes. Certaines solutions proposent également des outils de forensique (enregistrement de l'activité des containers et analyse post incident)

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Falco
- Sysdig Secure

c) Application des politiques de sécurité et conformité

Mise en œuvre des règles et des directives de sécurité spécifiques pour les conteneurs, garantissant leur conformité avec les normes et les réglementations en vigueur. Ce contrôle de conformité peut être appliqué par un « admission controller » en entrée de Kubernetes ou dans la chaine CI/CD à travers divers outils.

Certains outils s'identifient comme des CSPM (Cloud Security Posture Management) ou des KSPM (Kubernetes Security Posture Management)

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Open Policy Agent
- Conftest

d) Intégrité des images

Vérification de l'authenticité et de l'intégrité des images des conteneurs, afin de prévenir l'injection de code malveillant ou de logiciels compromis.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Notary
- Cosign

e) Logging, monitoring et auditing

Technologie d'enregistrement détaillé des activités des conteneurs, surveillance constante de leurs opérations et revue périodique des logs pour détecter les incidents de sécurité.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Stack ELK (Elasticsearch, Logstash, Kibana)
- Sysdig Monitor

3.5.7 BUG BOUNTY

Le Bug Bounty est une approche de cybersécurité qui consiste à offrir une rémunération financière ou d'autres incitations à des chercheurs en sécurité, considérés comme des «hackers éthiques», pour identifier et signaler des vulnérabilités ou des failles de sécurité dans les systèmes informatiques, les applications ou les plateformes d'une entreprise ou d'une organisation.

Les participants sont principalement les chercheurs en sécurité, qui peuvent être des individus professionnels, des étudiants ou même des passionnés. Ils sont responsables de trouver et de signaler les failles de sécurité dans les systèmes testés. Les organisations peuvent aussi inclure des membres de leur propre équipe de sécurité interne dans le programme.

Un programme de Bug Bounty s'appuie généralement sur une plateforme technologique qui permet d'outiller le processus de bout en bout. Ces plateformes permettent aux organisations de gérer et de suivre les rapports de vulnérabilités, d'organiser les programmes de récompenses, et de communiquer avec les chercheurs en sécurité. Elles offrent également des fonctionnalités de tri automatique des rapports, ainsi que des outils de communication et de collaboration.

La sélection même des chercheurs en sécurité peut se faire via des plateformes publiques de Bug Bounty. Ces plateformes disposent de grandes communautés de chercheurs en sécurité, dont certains ont des profils publics avec leurs compétences, des expériences et des réalisations précisées. Les organisations peuvent soit choisir des chercheurs spécifiques, soit permettre à n'importe quel chercheur inscrit sur la plateforme de participer. Certaines organisations peuvent également organiser des événements spécifiques, comme des hackathons ou des challenges, pour attirer des chercheurs.

Les systèmes pouvant faire l'objet de tests varient selon les organisations. Ils peuvent inclure des applications web, des sites mobiles, des API, des réseaux, des dispositifs IoT, des applications bureautiques, etc. L'objectif est de couvrir autant que possible les différents points d'entrée de l'organisation. Certaines organisations limitent les tests aux composants externes (c'est-à-dire accessibles sans authentification) pour minimiser les risques, tandis que d'autres autorisent les tests complets, y compris les zones internes. Les plateformes permettent de gérer et de diffuser les informations sur ces périmètres à adresser et les limites.

Les chercheurs en sécurité utilisent divers outils et techniques qui leur sont propres pour tester les systèmes. Cela peut inclure des analyses statiques et dynamiques, des tests manuels et automatiques, des attaques de type «social engineering», etc.

Après avoir identifié une vulnérabilité potentielle, les chercheurs la documentent minutieusement, fournissant généralement des informations sur la reproduction et la gravité du bug. La plateforme de Bug Bounty permet au chercheur de déclarer et documenter son travail. Ils y publient un rapport détaillé qui inclut une description de la vulnérabilité, des étapes de reproduction, et des recommandations de correction. Les organisations ont ensuite une période pour confirmer ou infirmer le bug. Si la vulnérabilité est confirmée, l'organisation travaille à sa correction. Une fois la correction apportée et validée, la récompense est versée au chercheur.

Les récompenses peuvent être financières, en nature, ou des reconnaissances publiques. Leur montant et leur forme dépendent de la gravité de la vulnérabilité, de la qualité du rapport, et des politiques de l'organisation. Certaines organisations proposent également des bonus pour les vulnérabilités particulièrement critiques ou pour les chercheurs qui ont contribué de manière exceptionnelle. Les plateformes de Bug Bounty permettent de gérer ces transactions.

Les plateformes de Bug Bounty permettent de gérer la communication entre l'organisation et les chercheurs de manière claire et régulière. Les chercheurs doivent être informés de l'état de leurs rapports, des correctifs mis en place, et des récompenses attribuées. La transparence est un aspect crucial pour maintenir la confiance et encourager les bonnes pratiques.

Il est important de noter que chaque programme de Bug Bounty peut avoir des modalités spécifiques, adaptées aux besoins et aux contraintes de l'organisation. Cependant, les principes de base restent similaires et visent à améliorer la sécurité en encourageant la collaboration entre les entreprises et les chercheurs en sécurité.

Les systèmes pouvant faire l'objet de tests varient selon les organisations. Ils peuvent inclure des applications web, des sites mobiles, des API, des réseaux, des dispositifs IoT, des applications bureautiques, etc. L'objectif est de couvrir autant que possible les différents points d'entrée de l'organisation. Certaines organisations limitent les tests aux composants externes (c'est-à-dire accessibles sans authentification) pour minimiser les risques, tandis que d'autres autorisent les tests complets, y compris les zones internes. Les plateformes permettent de gérer et de diffuser les informations sur ces périmètres à adresser et les limites.

Les chercheurs en sécurité utilisent divers outils et techniques qui leur sont propres pour tester les systèmes. Cela peut inclure des analyses statiques et dynamiques, des tests manuels et automatiques, des attaques de type «social engineering», etc. Après avoir identifié une vulnérabilité potentielle, les chercheurs la documentent minutieusement, fournissant généralement des informations sur la reproduction et la gravité du bug. La plateforme de Bug Bounty permet au chercheur de déclarer et documenter son travail. Ils y publient un rapport détaillé qui inclut une description de la vulnérabilité, des étapes de reproduction, et des recommandations de correction. Les organisations ont ensuite une période pour confirmer ou infirmer le bug. Si la vulnérabilité est confirmée, l'organisation travaille à sa correction. Une fois la correction apportée et validée, la récompense est versée au chercheur.

Très fréquemment, les organisations qui souhaitent développer un programme de bug bounty s'appuient sur des plateformes existantes mutualisées qui offrent le panel de services décrit précédemment et une communauté constituée de chercheurs.

Cependant, certaines organisations ont des contraintes fortes en matière de localisation du stockage et des traitements des données sensibles, telles que les directives du programme bug bounty et les rapports de vulnérabilités. S'il pourrait être envisageable d'héberger une plateforme de Bug Bounty propriétaire, nous n'avons pas identifié de solution logicielle commercial ou open-source pour ce faire. Cela impliquerait donc de construire et de maintenir un système interne pour gérer le programme, y compris la publication des règles du jeu, la gestion des chercheurs, la soumission des vulnérabilités, le tri et le suivi des rapports, ainsi que la gestion des récompenses. Outre le coût initial non négligeable (développement, installation, maintenance), l'inconvénient majeur reste la difficulté de constituer une communauté de chercheurs.

Pour répondre aux exigences de confidentialité des données, les gestionnaires des solutions intégrées peuvent proposer diverses approches, par exemple en installant des composants de la solution sur les serveurs de l'entreprise. Ces approches hybrides concilient l'avantage d'un service géré par l'éditeur avec le contrôle et la confidentialité d'une installation on-premise.

Pour une organisation qui souhaite initier une approche Bug Bounty, il est recommandé de disposer au préalable d'un niveau de maturité déjà assez avancé en matière de tests de sécurité et de remédiation. En effet, d'une part, des applications trop vulnérables peuvent entrainer des rapidement des couts importants en Bug Bounty.

Il apparait bien plus économique de détecter et traiter les vulnérabilités les plus évidentes avec des approches plus traditionnelles (SAST, DAST notamment). D'autre part, les équipes de développement doivent être suffisamment réactives pour analyser et corriger les vulnérabilités dès lors qu'elles sont documentées par des chercheurs. Une organisation de développement peu mature et peu réactive peut entrainer frustration et désintéressements au sein de la communauté de chercheurs.

Enfin, nous avons relevé au sein de la communauté d'intérêt des entreprises qui choisissent de s'appuyer sur ces plateformes pour réaliser des audits plus ponctuels d'applications. Par exemple, en cas d'incident de sécurité, les applications impactées sont soumises à un programme de bug bounty limité dans le temps afin d'obtenir un rapport sur la posture de sécurité associée et de définir les plans d'action adéquats pour l'améliorer.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Yogosha
- Yes We Hack
- HackerOne
- Bugcrowd
- Synack

3.6 APPLICATION PROTECTION

Différentes technologies permettent d'éviter l'exploitation de vulnérabilités alors qu'une application est en cours d'exécution. Il s'agit d'une ultime protection si des vulnérabilités n'ont pas pu être corrigées plus tôt dans le cycle de vie. Ce sont plutôt des technologies de prévention, ne reposant pas sur l'utilisation d'un scanner mais sur une analyse (et une réaction) en temps-réel, à l'exécution.

Elles s'intègrent complètement dans une démarche de cybersécurité agile, dans la mesure où elles permettent de réagir rapidement à une cybermenace sans attendre que les applications vulnérables ne soient corrigées : elles permettent notamment de mitiger les risques et des situations de crise.

Deux approches distinctes sont identifiées dans ce domaine : les firewalls applicatifs (WAF : Web Application Firewall) et l'auto-protection des application (RASP : Runtime Application Self-Protection).

3.6.1 WEB APPLICATION FIREWALL (WAF)

Le WAF (Web Application Firewall) se positionne comme pare-feu dédié à la protection des applications web. En amont de l'application dans le flux réseau, il permet notamment d'empêcher, l'exploitation d'une vulnérabilité en analysant le trafic HTTPS/S et bloquant des patterns d'attaques. Les requêtes qui s'avèrent malveillantes sont rejetées.

Traditionnellement, un WAF est configuré avec un ensemble de règles pour détecter des menaces dans les requêtes passées à l'application. Ces règles sont souvent basées sur une approche d'expressions régulières ou assimilée. Dans certains cas ; le WAF analyse aussi les réponses.

Les WAF peuvent être implémentés avec des solutions logicielles, parfois directement embarquées dans des les serveurs web, ou des appareils physiques dédiés.

Plusieurs limites sont communément constatées avec l'approche WAF :

- Le dispositif bloque toute requête présentant des caractéristiques malveillantes sans savoir si l'application visée aurait été effectivement vulnérable. Fort heureusement, il n'est pas exclu qu'une application ne soit pas vulnérable à un type d'attaque donné. En conséquence, les logs des WAF listent une quantité très importante de blocages sans que ceux-ci ne soient une réelle indication quant au niveau de sécurité de l'application, ces logs ne requièrent pas d'action à proprement parler, notamment de la part des développeurs.

- Il y a souvent des faux-positifs : des requêtes légitimes qui sont bloquées car identifiées comme malveillantes par les règles définies. Dans ce cas, les utilisateurs de l'application sont impactés et voient leurs transactions échouer. Il est alors nécessaire d'affiner les règles ou de mettre en place des exclusions. Le plus souvent, les règles problématiques sont simplement désactivées et ne protègent alors plus des cas d'attaques effectifs.

- En règle générale et traditionnellement, le dispositif de WAF est positionné en entrée de DMZ de manière analogue aux firewalls. Ainsi, l'ensemble du trafic web est analysé et plusieurs applications sont protégées. Il est alors assez souvent difficile de distinguer les exclusions en fonction des applications, et, assez fréquemment, une règle désactivée car problématique pour une application donnée l'est pour un ensemble plus large du périmètre. Dans le temps, il est complexe pour les organisations de revalider régulièrement la nécessité de maintenir les exclusions en place. Il s'en suit, au grès des évolutions des applications, des nouveaux usages et des décommissionnements, une perte de maitrise des jeux de règles qui se trouvent fortement personnalisés avec une couverture de risques finalement assez limitée. Ces jeux de règles complexes aux multiples exclusions sont aussi un frein à l'évolution des WAF, les organisations craignant un processus de migration long, couteux et risqué.

Pour pallier certaines de ces limites, les fournisseurs ont développé des offres avancées dites de WAF nouvelle génération ou 'Next Gen'. Ces offres s'adossent généralement à des réseaux de diffusion de contenus (CDN : Content Delivery Network) qui permettre, en outre, de mitiger les risques d'attaques DDoS. Dans ces modèles, le dispositif de protection est délocalisé chez le fournisseur et, le plus souvent, mutualisé avec l'ensemble des clients. Il s'agit d'une approche « Cloud ».

Ces solutions de nouvelle génération se distinguent par l'intégration d'une veille sur les menaces (Threat Intelligence). L'analyse ne se limite plus à analyser le contenu des requêtes au cas par cas mais considère le contexte global de l'activité et du trafic du demandeur. Ainsi, des activités malicieuses vont être détectées et bloquées sur la base d'une analyse du comportement général d'un acteur constaté sur Internet pour un ensemble d'organisation. Les blocages « faux-positifs » sont drastiquement réduits, il en va par conséquence de même pour les exclusions et la maintenance de celles-ci.

Outre le fait que l'approche « Next gen » ait une couverture de risques plus étendue qu'un WAF traditionnel, celle-ci s'avère moins coûteuse en termes de maintenance et plus fiable par le nombre réduit d'exclusions.

Enfin, il est à noter que certaines organisations examinent aussi la possibilité de distribuer le WAF sur le périmètre applicatif en ayant des WAF unitaires placés en frontal de chaque application. Les technologies Cloud, notamment container et IaC (Infrastructure as Code) facilite cette mise en place en automatisant le déploiement et la configuration des multiples WAFs. L'objectif de ce dispositif est d'éviter la mutualisation des WAF et ainsi de faciliter la maitrise des exclusions dans les jeux de règles.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- ModSecurity
- Datadome

3.6.2 RUNTIME APPLICATION SELF-PROTECTION (RASP)

Les technologies identifiées sous l'acronyme RASP (Runtime Application Self-Protection) visent à ce que l'application en cours d'exécution bloquent automatiquement les attaques en cas d'exploitation d'une vulnérabilité. Ces mécanismes de protection s'intègrent directement à l'application et/ou à l'environnement technique d'exécution.

L'approche la plus commune pour le RASP consiste à exploiter les capacités d'instrumentation de l'application fournies par les moteurs d'exécution tels que .NET framework ou la JVM java. Ces capacités d'instrumentation permettent de suivre précisément l'exécution de l'application pendant le traitement d'une transaction.

En cas d'exploitation d'une vulnérabilité, il est possible d'identifier le comportement déviant de l'application et de bloquer l'exécution en question. En règle générale, seule la transaction frauduleuse détectée est bloquée, les autres utilisateurs légitimes de l'application ne sont pas affectés.

L'approche RASP diffère fondamentalement d'une protection par WAF par le fait qu'elle n'intervient qu'en cas d'exploitation effective d'une vulnérabilité. Là ou un WAF bloquerait toute requête malveillante avant que celle-ci n'atteigne l'application, le RASP ne va faire échouer l'exécution que si l'application est effectivement vulnérable, autrement dit, l'attaque aurait été réussie. Dans cette situation, il est alors opportun d'informer l'équipe de développement de la présence effective d'une vulnérabilité permettant de s'inscrire dans une démarche d'amélioration continue et de défense en profondeur. Pour la même raison, l'approche RASP présente des taux de faux-positifs extrêmement bas voir nuls.

Si l'approche RASP présente des avantages forts, il y a néanmoins certains inconvénients à considérer :

- L'instrumentation de l'application nécessite des composants additionnels en production, intégrés à l'environnement d'exécution, et présentant donc des risques supplémentaires pour la performance voire la disponibilité. En pratique, les responsables de production restent souvent réticents à déployer des solutions fortement intégrées à la production, notamment dans les environnements sensibles et fortement sollicités. Les éditeurs des solutions de RASP implémentent des stratégies variées pour réduire l'empreinte des agents sur la production. Il faut noter aussi que certains éditeurs intègrent les capacités de sécurité (RASP) dans leurs solutions globales de gestion de la performance applicative, ce qui permet de mutualiser l'impact sur la production pour plusieurs usages favorisant ainsi l'adoption.

- Les risques couverts par la solution RASP ne sont pas équivalents à ceux d'un WAF. En effet, les WAF implémentent certaines règles qui ont plus trait à des vulnérabilités protocolaires et réseau que des vulnérabilités dans le code applicatif. Il semble donc opportun de conserver des WAF pour ces aspects complémentaires. En revanche, la cohabitation entre WAF est RASP n'est pas aisée. Si les deux solutions peuvent sans soucis protéger en même temps, si un WAF bloque des requêtes malveillantes qui auraient exploité une vulnérabilité de l'application, on perd l'avantage du RASP de détecter ces exploitations et de pouvoir corriger les vulnérabilités. De même, dans le cas de requêtes légitimes, si elles sont bloquées par le WAF, on perd l'avantage du RASP d'être moins sensible aux faux-positifs. En définitive, les organisations s'accordent sur le fait qu'il faille désactiver les règles WAF qui correspondent à des risques couverts par le RASP. Dans la pratique, ce n'est pas évident.

- Enfin, et non des moindres, la technologie RASP demeure assez peu adoptée pour l'instant par les organisations, au profit, vraisemblablement des WAF de nouvelle génération. Cela a amené ces dernières années, le cabinet Gartner à classer le RASP comme obsolète. Une adoption du RASP à grande échelle dans l'entreprise présente donc un risque d'obsolescence et de défaut de maintenance de la solution dans le temps qui doit être pris en compte.

Il est à noter quelques usages particuliers de certaines solutions RASP qui méritent attention. Certaines solutions permettent de palier à des défauts d'obsolescence des environnements d'exécution qui ne seraient plus maintenus. Par exemple, des applications développées pour des piles java obsolètes pourraient continuer d'être exécutées avec un risque maitrisé. Cette approche peu présenter un intérêt économique dans certaines situations où la migration des applications nécessite des travaux de réécriture et d'ingénierie importants.

Parmi les outils couvrant cette technologie, peuvent être cités à titre d'exemple :

- Sqreen RASP
- Contrast RASP
- Licel Dexprotector (mobile)

< Studio des Communs >

CAMPUS CYBER

POUR EN SAVOIR PLUS : **WIKI.CAMPUSCYBER.FR**
ADRESSE MAIL DE CONTACT : **COMMUNAUTES@CAMPUSCYBER.FR**
5 - 7 RUE BELLINI 92800, PUTEAUX

Ce projet a été financé par le gouvernement dans le cadre du Programme d'investissements d'avenir.