AXOPEN

La cybersécurité dans le développement logiciel

Dossier Technique

Sommaire

AXOPEN
235 cours Lafayette – 69006 LYON
+33 (0)4 82 53 26 44 - contact@axopen.com
www.axopen.com

SAS au capital de 127 200 euros - SIREN 494 526 908 - RCS LYON
TVA Intra : FR74494526908 - APE: 6201Z

AXOPEN
235 cours Lafayette – 69006 LYON
+33 (0)4 82 53 26 44 - contact@axopen.com
www.axopen.com

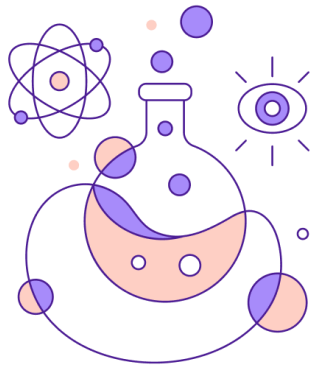
SAS au capital de 127 200 euros - SIREN 494 526 908 - RCS LYON
TVA Intra : FR74494526908 - APE: 6201Z

Édito

Auparavant confinées à leur infrastructure interne, les applications métiers se sont progressivement ouvertes sur l'extérieur. Il n'est aujourd'hui plus possible de se reposer uniquement sur la couche réseau pour garantir la sécurité du système d'information.

Les applications sont désormais directement exposées aux attaques, ce qui ramène les enjeux de sécurité au niveau de la couche applicative, et donc du code. Ce rôle, autrefois porté exclusivement par les équipes infrastructures, doit désormais être intégré dès le développement.

À cela s'ajoute l'expansion du système d'information et l'augmentation de sa complexité ces dernières années, ce qui accroît mécaniquement la surface d'attaque.



Les menaces sont bien réelles, mais les moyens de protection sont nombreux. Même s'il est impossible d'éliminer totalement les risques, certaines règles doivent impérativement être respectées pour ralentir, voire empêcher, les tentatives d'intrusion. C'est ce que nous allons aborder dans ce dossier technique.

AXOPEN
235 cours Lafayette – 69006 LYON
+33 (0)4 82 53 26 44 - contact@axopen.com
www.axopen.com

SAS au capital de 127 200 euros - SIREN 494 526 908 - RCS LYON
TVA Intra : FR74494526908 - APE: 6201Z

I. Les principales menaces

La cybersécurité du code constitue aujourd'hui un enjeu majeur. Les vulnérabilités applicatives représentent désormais l'un des principaux vecteurs d'attaque, devant les failles d'infrastructure traditionnelles.

1.1. Les vulnérabilités courantes

Les applications web sont naturellement les plus exposées aux attaques classiques. Parmi celles-ci, on retrouve notamment les failles listées dans l'OWASP Top 10 [1], telles que les injections SQL, les injections XSS ou les vulnérabilités CSRF, qui exploitent la confiance accordée aux utilisateurs authentifiés pour exécuter des actions non autorisées.

Ces failles trouvent leur origine dans des pratiques de développement insuffisamment sécurisées : absence de validation des données en entrée, mauvaise gestion des sessions, ou encore manque de chiffrement des données sensibles. La complexité croissante des applications et la pression sur les délais de livraison accentuent ces risques.

1.2. Les composants tiers

Les failles ne proviennent pas toujours du code développé en interne. En effet, l'utilisation de composants tiers peut introduire des vulnérabilités si elle n'est pas rigoureusement encadrée. Le choix de ces composants est donc déterminant, et la confiance qui leur est accordée doit faire l'objet d'une analyse approfondie.

Plusieurs bibliothèques open source ont déjà été compromises, soit volontairement par des développeurs malveillants, soit involontairement par l'ajout de code vulnérable. Parmi les exemples notables, on peut citer la faille Log4Shell qui, en décembre 2021[2], qui a exposé des millions d'applications à l'exécution de code à distance via une simple manipulation de message de log. Ou encore la faille Heartbleed, découverte en 2014 dans OpenSSL [3], qui permettait l'extraction de clés privées et de données sensibles sur des millions de serveurs web.

[1] https://www.axopen.com/blog/2024/11/owasp-top-10-failles-api/
[2] https://www.cert.ssi.gouv.fr/alerte/CERTFR-2021-ALE-022/
[3] https://en.wikipedia.org/wiki/Heartbleed

Une veille active est donc essentielle pour sécuriser ces composants : plus ils sont utilisés et maintenus par une communauté sérieuse, plus ils sont audités - mais cela en fait aussi des cibles de choix. Plus un composant est répandu, plus il est rentable à attaquer.

Le même constat s'applique au CMS WordPress, longtemps perçu comme peu sécurisé. En réalité, la plupart des vulnérabilités proviennent de l'absence de mises à jour. Utilisé par plus de 40 % des sites web dans le monde[4], WordPress est une cible privilégiée pour les attaquants.

1.3. Impacts et conséquences

Ce principe du "plus utilisé, plus attaqué" se retrouve aussi dans les développements spécifiques. Un logiciel sur-mesure, par définition moins diffusé, sera moins ciblé qu'une solution SaaS populaire. Cela ne signifie pas pour autant qu'il est plus sécurisé. Ce n'est pas parce qu'une entreprise ne constitue pas une "grosse cible" qu'elle est à l'abri. Si un attaquant y voit un intérêt, il mettra les moyens nécessaires pour y accéder.

Un développement spécifique doit bénéficier du même niveau d'exigence en matière de sécurité qu'une application à grande échelle. Il mérite même une vigilance renforcée, car les retours utilisateurs sont moins nombreux, et les failles peuvent rester invisibles plus longtemps.

Au-delà des conséquences techniques, une attaque porte atteinte à la réputation de l'entreprise, souvent plus durablement que l'incident lui-même. Les failles applicatives peuvent provoquer des fuites de données, des divulgations d'informations sensibles, y compris vis-à-vis de collaborateurs internes qui n'auraient pas dû y avoir accès. Et, restaurer la confiance après un tel incident est toujours difficile.

[4] https://w3techs.com/technologies/overview/content_management

Dans ce document, nous allons passer en revue les différentes phases de développement d'un projet afin d'identifier les bonnes pratiques à adopter pour éviter de laisser des portes ouvertes aux attaquants.

II. Sécurité lors de la phase de conception

La phase de conception est une étape clé pour intégrer la sécurité dès le début d'un projet de développement logiciel. Cette approche, appelée Security by Design, vise à anticiper les problèmes de sécurité en les prenant en compte dès la conception, plutôt que de les corriger a posteriori - une démarche souvent plus coûteuse et moins efficace.

2.1 Modélisation des menaces et architecture sécurisée

Avant même d'écrire la première ligne de code, il est essentiel de réaliser une modélisation des menaces.

Ce processus consiste à identifier méthodiquement les éléments à protéger, les vecteurs d'attaque potentiels et à évaluer les risques associés. Des méthodologies structurées comme STRIDE [5] (*Spoofing, Tampering, Repudiation, Information Disclosure, Denial of service, Elevation of privilege*) offrent un cadre analytique pour cette étape fondamentale.

Cette analyse permet d'orienter les décisions architecturales et de définir les stratégies de protection adaptées aux menaces identifiées.

L'architecture doit ensuite être conçue avec plusieurs niveaux de protection :

- Segmentation : Séparer les différentes parties de l'application pour limiter la propagation d'une éventuelle compromission.
- Defense in depth : Multiplier les barrières de sécurité pour qu'une seule faille ne compromette pas tout le système.
- Fail secure : En cas de défaillance, le système doit adopter l'état le plus sécurisé par défaut.

[5] https://web.archive.org/web/20070303103639/http://msdn.microsoft.com/msdnmag/issues/06/11/ThreatModeling/default.aspx

2.2 Sélection judicieuse des composants et bibliothèques

Le choix des composants externes est déterminant pour la sécurité globale de l'application.

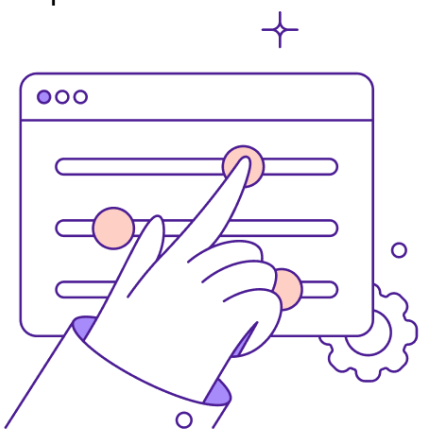
En matière d'authentification et de gestion des identités, il est recommandé de privilégier des solutions éprouvées plutôt que de développer sa propre implémentation, afin de limiter les risques.

L'intégration de l'authentification multifacteur (MFA), ainsi que l'usage de protocoles standardisés comme OAuth 2.0, OpenID Connect ou SAML, constitue une base solide.

La gestion des sessions doit également être pensée avec soin, en incluant des mécanismes d'expiration appropriés selon la sensibilité des données manipulées.

Concernant la protection des données sensibles, plusieurs points sont à prendre en compte :

- Définir une politique claire de classification des données.
- Choisir des algorithmes de chiffrement robustes et à jour (AES, RSA avec des tailles de clés appropriées).
- Utiliser des fonctions de hachage sécurisées pour les mots de passe (Argon2, bcrypt, PBKDF2).
- Ne jamais stocker de données sensibles en clair, y compris dans les fichiers de configuration.



2.3 Le principe du moindre privilège, fondement de la sécurité

Le principe du moindre privilège (Least Privilege) constitue l'un des piliers de la sécurité applicative.
Il stipule que chaque composant du système ne doit disposer que des droits strictement nécessaires à l'accomplissement de sa fonction.

Cette approche restrictive doit s'appliquer à tous les niveaux :

- Les comptes de service applicatifs doivent avoir des droits limités.
- Les accès aux bases de données doivent être restreints aux opérations nécessaires.
- Les communications entre services doivent être filtrées selon les besoins réels.

Ce principe s'étend également à la conception des API, qui doivent intégrer des contrôles d'accès granulaires, ainsi que réduire la surface d'attaque au maximum.

Chaque API doit suivre 3 concepts clés :

- Confidentialité : seul un utilisateur authentifié et avec les bons rôles doit pouvoir accéder aux ressources de votre API.
- Intégrité : s'assurer que les données ne soient pas modifiées ou manipulées par des utilisateurs non autorisés
- Disponibilité : maintenir l'accès fiable et continu aux services pour les utilisateurs légitimes

2.4 Autres mécanismes de protection

La mise en place de quotas et de limitations de débit contribue à prévenir les abus et les attaques par déni de service (DoS).

Un système de journalisation complet des accès et des actions permet de détecter plus rapidement les comportements suspects et facilite l'investigation en cas d'incident de sécurité.

III. Phase de développement

La sécurité d'une application pendant son développent est cruciale, notamment au niveau du code, où chaque détail compte.
Il est essentiel de veiller à la fois à la sécurité technique, qui couvre des aspects tels que la prévention des injections SQL ou l'utilisation de protocoles d'authentification robustes, et à la sécurité logique de l'application, qui vise à s'assurer que les utilisateurs ne peuvent effectuer que les actions prévues.

Par exemple : vérifier qu'un paiement a bien été validé avant de déclencher la livraison d'un produit. Ces principes s'appliquent également aux processus métiers.

Dans cette section, nous nous concentrerons sur deux axes essentiels :

- La validation des entrées
- La gestion des rôles et des autorisations

Nous évoquerons également les normes et référentiels utiles pour renforcer la sécurité du code.

3.1 Validation des entrées

La validation des entrées d'une API est cruciale tant pour la sécurité que pour le fonctionnement correct de l'application. Il est essentiel de s'assurer que les valeurs attendues sont présentes dans les attributs et qu'il est impossible d'altérer le comportement prévu de l'application par l'intermédiaire de ces paramètres.

Cette vérification devient encore plus cruciale lorsque les entrées utilisateur permettent de générer des requêtes SQL ou des appels vers d'autres API. L'utilisation de requêtes préparées et de mécanismes sécurisés permet d'éviter des attaques telles que l'injection SQL ou les attaques par rebond (SSRF - Server Side Request Forgery).

3.2 Gestion des rôles et autorisations

Il est essentiel de vérifier systématiquement le contexte de sécurité au niveau du code et de s'assurer que le mécanisme de gestion des rôles utilisateur est fiable.
La récupération et l'attribution des rôles doivent reposer sur des sources de confiance, comme un JWT émis par un fournisseur de jetons sécurisé. À l'inverse, il ne faut jamais se baser sur des sources non sécurisées, telles que le local storage, car un utilisateur malveillant pourrait en modifier le contenu s'il a connaissance du secret du jeton.

Une fois l'authenticité du rôle utilisateur garantie, il est impératif de le vérifier avant chaque opération critique dans l'application métier.

3.3 Protection des données

La protection des données, tant lors de leur stockage que de leur transmission, est un autre aspect clé. Les données sensibles doivent être chiffrées en suivant les recommandations de la RGPD. Il est également important de s'assurer qu'il n'est pas possible de récupérer facilement ces données à travers l'API. Globalement, les données sensibles doivent souvent être associées à un mécanisme de sécurité d'appartenance. C'est-à-dire qu'il faut s'assurer que l'utilisateur est bien propriétaire de la donnée avant de lui transmettre. Dans ce cas précis, ce n'est plus le rôle de l'utilisateur qui importe, mais son identité.

3.4 L'OWASP Top 10 et les référentiels de sécurité

L'OWASP Top 10[6] constitue une excellente base pour évaluer le niveau de sécurité d'une application.

Cette liste identifie les vulnérabilités les plus critiques à surveiller et fournit un cadre utile pour s'assurer que les principaux risques sont couverts.
Bien que ces recommandations soient souvent génériques, elles offrent une structure précieuse pour auditer et renforcer la sécurité du code.

[6] https://owasp.org/www-project-top-ten/

D’autres référentiels et normes peuvent compléter cette approche, notamment :

- ISO/IEC 27001 (gestion de la sécurité de l’information)
- ISO/IEC 27034 (sécurité des applications)
- Le NIST Secure Software Development Framework (SSDF)[7]
- Le RGPD, pour les aspects liés à la protection des données personnelles

Ces standards permettent de structurer une démarche globale et rigoureuse en matière de sécurité logicielle.

[7] https://csrc.nist.gov/projects/ssdf

IV. Sécurité lors de la phase d'intégration

Malgré toute la bonne volonté que l'on peut mettre dans un projet, il n'est pas rare de passer à côté de certains aspects de sécurité, par oubli ou par méconnaissance.

Aujourd'hui, l'intégration et le déploiement continu (CI/CD) sont devenus des pratiques courantes. Souvent centrés sur la compilation et la livraison de l'application, ces processus s'inscrivent dans une démarche DevOps.

De la même manière qu'on automatise les déploiements pour les fiabiliser, il est possible d'automatiser les tests de sécurité tout au long du cycle de vie du projet. C'est ce que l'on appelle le DevSecOps.

Le DevSecOps repose sur une chaîne d'outils permettant de réaliser différentes analyses, notamment :

- L'analyse statique du code,
- L'analyse des dépendances,
- Le scanning dynamique de l'application en cours d'exécution pour détecter d'éventuelles vulnérabilités.

4.1 L'analyse de sécurité statique (SAST)

L'objectif de l'analyse SAST (*Static Application Security Testing*) est de détecter des vulnérabilités en analysant uniquement le code source, à l'aide de règles plus ou moins complexes.

Bien configurée, cette analyse permet de détecter efficacement des failles telles que les injections SQL ou XSS, la présence de secrets en clair, ou encore des accès à des données non vérifiés.

Une fois l'analyse terminée, un rapport détaillé est généré, listant les règles concernées ainsi que leur emplacement dans le projet. Cela facilite la correction et dans la plupart des cas, l'outil fournit même des recommandations précises sur la manière de corriger chaque point détecté.

L'un des avantages du SAST est qu'il ne nécessite pas d'exécuter l'application, ce qui le rend relativement simple à intégrer dans une chaîne CI/CD. La principale difficulté réside dans la qualité des règles définies, ainsi que dans la prise en charge des multiples langages et frameworks utilisés.

C'est pourquoi cette approche est souvent mise en œuvre à l'aide de solutions éprouvées du marché, prêtes à l'emploi.

Parmi les outils les plus connus, on retrouve SonarQube, qui prend en charge de nombreux langages et propose des milliers de règles liées à la sécurité, mais aussi aux bonnes pratiques de développement.

4.2 L'analyse de sécurité dynamique (DAST)

L'analyse DAST (*Dynamic Application Security Testing*) vise à détecter les vulnérabilités en exécutant l'application et en testant l'exploitation de failles connues, telles que les injections, le « path traversal »[8], ou encore les vulnérabilités listées dans l'OWASP Top 10.

Une fois l'analyse terminée, un rapport est généré, listant les points d'entrée vulnérables ainsi que les failles détectées.
Il appartient ensuite à l'équipe de développement d'identifier la cause racine et d'apporter les correctifs nécessaires.

C'est l'une des analyses les plus complexes à mettre en œuvre, car elle nécessite non seulement que l'application soit fonctionnelle et accessible, mais aussi que l'environnement soit le plus proche possible de la production. Par exemple, un test d'injection SQL ne sera pertinent que si l'application est réellement connectée à une base de données.

Il est important de ne pas lancer ces tests sur l'environnement de production. L'objectif du DAST étant de vérifier l'existence réelle de failles en tentant de les exploiter, cela peut entraîner une instabilité de l'environnement analysé, voire des pertes de données.

Parmi les outils les plus connus pour réaliser ce type d'analyse, on peut citer OWASP ZAP, une solution open source largement utilisée dans les phases de test de sécurité dynamique.

[8] https://en.wikipedia.org/wiki/Directory_traversal_attack

4.3 L'analyse de la composition logicielle (SCA)

L'analyse SCA (*Software Composition Analysis*) permet d'identifier les failles connues dans les dépendances utilisées par un projet.

Pour cela, elle s'appuie sur des bases de données publiques de vulnérabilités (comme la CVE – Common Vulnerabilities and Exposures).

C'est une analyse particulièrement utile, car la quasi-totalité des projets logiciels intègrent des centaines, voire des milliers de dépendances tierces. Souvent, rien que le fait d'utiliser une version obsolète d'un langage ou d'un framework peut introduire des failles de sécurité.

Une fois l'analyse effectuée, un rapport est généré. Il contient :

- La liste des dépendances vulnérables,
- Les détails des failles associées,
- Leur niveau de criticité,
- La version corrigée disponible, si elle existe,
- Et, dans certains cas, les modes d'exploitation connus.

La correction est plus ou moins simple selon la dépendance concernée et la disponibilité d'une mise à jour.
Mais ces failles doivent être traitées, car si elles figurent dans des bases publiques, c'est qu'elles sont connues des attaquants et potentiellement exploitées activement.

Cette analyse est relativement facile à mettre en place, car elle repose principalement sur la liste des bibliothèques utilisées par le projet.

Parmi les outils les plus utilisés, on retrouve :

- Dependency-Track
- Dependabot
- Snyk
- Trivy ou Grype (pour les dépendances au sein de conteneurs)
- Et Renovate, qui permet d'automatiser la mise à jour des dépendances tout au long du cycle de vie du projet.

4.4 La détection des secrets (Secret detection)

Comme son nom l'indique, la détection des secrets vise à identifier d'éventuelles informations sensibles (clés d'API, tokens, mots de passe, certificats, etc.) présentes dans le code source, afin de les supprimer et de les sécuriser correctement.

La mise en place de cette analyse est relativement simple, mais sa correction peut s'avérer complexe. Elle peut nécessiter de modifier l'historique Git pour purger les secrets, mais aussi de revoir l'infrastructure du projet, notamment la manière dont les secrets sont stockés et rendus accessibles. Cette gestion doit être sécurisée tout au long du cycle de vie du projet, aussi bien dans la chaîne d'intégration continue qu'en production.

Ce type d'analyse est particulièrement important sur les projets open source ou accessibles publiquement. Des secrets exposés peuvent donner accès à des services ou à des données sensibles. C'est une faille critique, d'autant plus que de nombreux pirates automatisent aujourd'hui la recherche de secrets sur des plateformes comme GitHub pour tenter de les exploiter.

Une fois l'analyse réalisée, on obtient un rapport listant les secrets détectés ainsi que leur localisation dans le projet. Il faut alors révoquer les secrets exposés, en générer de nouveaux et les stocker dans un endroit sécurisé.

Pour cela, on peut utiliser un coffre-fort de secrets qui permet de les stocker de manière sécurisée, de les chiffrer et parfois même de gérer leur rotation automatique.

Parmi les outils d'analyse, on peut citer ceux fournis par GitHub ou GitLab, ainsi que Gitleaks. Pour le stockage sécurisé, on retrouve des solutions comme HashiCorp Vault, AWS Secrets Manager ou Azure Key Vault.

V. La sécurité applicative : une préoccupation long terme !

Une fois les différentes phases du projet passées et l'application mise en production, la sécurité ne s'arrête pas là. Au contraire, c'est à ce moment que l'application devient réellement exposée. Si elle doit être attaquée, c'est maintenant que ça se jouera.

5.1 Savoir que l'on est attaqué : l'observabilité rentre en jeu

Se faire attaquer, ce n'est jamais une bonne nouvelle, mais le pire, c'est de ne même pas s'en rendre compte. Imaginez que des données soient corrompues ou volées sans que vous en ayez la moindre idée, ni aucun moyen de le prouver… c'est exactement ce qu'il faut éviter à tout prix.

C'est pourquoi il est essentiel de mettre en place un système d'observabilité. Concrètement, cela signifie centraliser la collecte des logs, des métriques et des traces, puis les analyser pour repérer tout comportement suspect. Ce système permet aussi de détecter des signaux faibles qui pourraient annoncer une attaque en préparation.

Les alertes sont là pour prévenir rapidement les personnes en charge, qui pourront alors réagir au plus vite. En plus, garder un historique des logs permet d'analyser ce qui s'est passé, pour mieux prévenir la prochaine fois.

Parmi les outils les plus utilisés, on trouve ELK, Splunk, AWS CloudWatch ou Azure Monitor.

Pour les environnements les plus sensibles, il existe des solutions plus sophistiquées, appelées SIEM (Security Information and Event Management). Ces systèmes collectent, centralisent, analysent et croisent les données de différentes sources pour détecter en temps réel les menaces ou incidents.

Ils vont même plus loin en analysant les comportements et les accès aux données, souvent avec l'aide de l'intelligence artificielle. Ce sont des outils puissants, parfois complexes à déployer, mais indispensables pour assurer la sécurité des infrastructures critiques.

5.2 Limiter la surface d'attaque

On parle souvent de surface d'attaque quand on aborde la sécurité, et cette surface ne cesse de s'élargir au fil des années, rendant son contrôle de plus en plus complexe. Aujourd'hui, les applications s'appuient sur de nombreux services managés, souvent pour de bonnes raisons, comme apporter plus de fonctionnalités ou de performance. Mais ces services doivent aussi être sécurisés.

La configuration de la sécurité pour chaque service chez les fournisseurs cloud est souvent un vrai casse-tête, mais c'est crucial pour éviter toute intrusion facile. Les autorisations doivent toujours être les plus restrictives possible. Oubliez l'idée de "tout ouvrir" parce que vous ne savez pas comment bien configurer les droits.

Pour limiter les attaques et bloquer les comportements suspects, une solution simple et efficace est d'utiliser un WAF (Web Application Firewall). Placé en première ligne devant votre application, il filtre le trafic et analyse les requêtes en temps réel. Grâce à des modèles de détection, il peut identifier et bloquer des attaques courantes comme les injections SQL, le cross-site scripting (XSS) ou les tentatives de path traversal. C'est une solution simple à mettre en place, peu coûteuse, et qui demande juste à être régulièrement mise à jour avec les dernières signatures d'attaque.

5.3 Auditer pour mieux anticiper

Nous avons parlé de systèmes externes pour sécuriser notre applicatif, mais le cœur reste dans le code, qui est le premier vecteur de risque. Au fil des évolutions et des corrections que subit l'application, des failles de sécurité peuvent apparaître. Pendant la maintenance, l'attention portée à la sécurité peut diminuer : face à la pression métier, il faut souvent traiter les corrections rapidement, et la sécurité devient alors le cadet des soucis des développeurs. Le turnover des développeurs peut aussi conduire à des raccourcis risqués. Plus globalement, un manque de sensibilisation des développeurs à la sécurité peut avoir des conséquences graves.

Le meilleur moyen de parer à ces risques est d'auditer régulièrement le projet. De préférence, cet audit doit être réalisé par un acteur externe ou une équipe indépendante, qui apportera un regard neuf et considérera la sécurité comme une priorité. En effet, l'audit de code complète l'audit classique de cybersécurité, qui se concentre davantage sur l'infrastructure que sur la partie métier. Par exemple, un audit cybersécurité pourra révéler une version obsolète d'un serveur web, tandis qu'un audit de code démontrera qu'il est possible de récupérer tous les comptes clients alors que ce n'est pas censé être possible.

VI. Et l'IA dans tout ça ?

L'utilisation de l'intelligence artificielle dans le développement logiciel ouvre des opportunités majeures, mais apporte aussi des défis réels en matière de sécurité.

6.1 Risques potentiels liés à l'IA

Des outils comme GitHub Copilot ou ChatGPT permettent de gagner du temps en générant automatiquement du code. Cependant, leur utilisation peut aussi introduire des risques importants. Les modèles d'IA étant souvent opaques, il est difficile de vérifier rapidement si le code produit est sécurisé. Sans une vérification rigoureuse par les développeurs, ces outils peuvent intégrer des vulnérabilités cachées ou reproduire de mauvaises pratiques.

Par ailleurs, l'analyse automatique du code par des plateformes externes peut exposer des données confidentielles à des risques de fuite. Il est donc essentiel de contrôler strictement l'usage de ces outils.

6.2 Nouveaux types d'attaques

Avec l'essor de l'IA, de nouvelles vulnérabilités spécifiques apparaissent :

- Prompt Injection : il s'agit de manipuler les instructions envoyées à l'IA pour influencer ses réponses. Les applications qui utilisent ces réponses sans vérification sérieuse deviennent vulnérables.
- Data Poisoning : contamination des données d'entraînement des modèles d'IA, ce qui peut provoquer des comportements indésirables ou dangereux dans les applications.

6.3 Aides concrètes apportées par l'IA

Malgré ces risques, l'IA peut être une aide précieuse pour la sécurité :

- Elle permet de détecter rapidement des comportements suspects ou des tentatives d'attaque grâce à une analyse automatisée approfondie.
- Elle facilite la revue de code en repérant efficacement des erreurs ou mauvaises pratiques souvent difficiles à détecter lors d'audits manuels.

En résumé, même si l'IA est un outil très puissant, elle doit être utilisée avec prudence et toujours accompagnée d'un contrôle humain vigilant pour garantir la sécurité des applications.

Conclusion : la sécurité, un défi permanent

La sécurité applicative est un processus continu qui doit être intégré à chaque étape du développement. Même avec les meilleures pratiques et outils, rien ne remplace l'humain : la vigilance, la remise en question et la formation régulière des équipes restent indispensables.

N'essayez pas de viser la perfection en matière de sécurité, c'est souvent le meilleur moyen de ne rien faire. Vous pouvez faire tous les efforts possibles, mais si un attaquant motivé et compétent veut vous cibler, il trouvera un moyen. L'objectif n'est pas une protection absolue, mais de rendre les attaques les plus difficiles et coûteuses possible, pour décourager les attaquants, tout en étant prêt à réagir rapidement en cas d'incident.

Annexes

Ce qu'il faut retenir

La cybersécurité applicative n'est plus une option. Une application mal sécurisée est une porte ouverte dans le SI et représente un risque majeur pour une entreprise. C'est pour cette raison que la sécurité applicative doit être intégrée dès le début d'un projet et rester une préoccupation tout au long de sa vie.

L'objectif n'est pas d'être invulnérable, car c'est impossible, mais de rendre toute attaque suffisamment couteuse et complexe pour décourager l'attaquant. Et si une brèche est découverte, il faut pouvoir la corriger rapidement.

Pour cela, dès la conception, les choix techniques doivent prendre en compte la sécurité, via le choix de composants fiables et maintenus. Il faut aussi imaginer les menaces auxquelles on s'expose et appliquer le principe de moindre privilège pour correctement segmenter l'architecture logicielle.

Ensuite, durant la phase de développement, il est essentiel de valider systématiquement les saisies utilisateurs, de gérer correctement les droits et les sessions, et de chiffrer les données sensibles.
Sans ces protections, des actions non autorisés peuvent mener à des pertes ou à des vols de données.

Comme tout sujet nécessitant de la rigueur, des failles peuvent facilement passer inaperçues. Il est donc conseillé de mettre en place des outils d'analyse automatisés, tel que des analyses de code source, de vérification des dépendances et de détection d'exposition de secrets.

Une fois l'application en production, la sécurité doit rester une priorité. De nouvelles vulnérabilités apparaissent chaque jour : une application sûr aujourd'hui peut devenir vulnérable demain.
C'est pourquoi, Il est essentiel de mettre en place de l'observabilité, des alertes et, si nécessaire ; un SIEM pour être informé rapidement d'une brèche de sécurité.

L'ajout d'un WAF peut également aider à réduire la surface d'attaque en se bloquant les menaces les plus connues.

Enfin, des audits réguliers permettent de vérifier que le système reste sécurisé au fil de ses évolutions.

L’IA peut être un allié puissant, mais elle doit être utilisée avec prudence. Le code qu’elle génère doit toujours être revu pour éviter d’introduire de nouvelles vulnérabilités.

Glossaire

AES (Advanced Encryption Standard)
Algorithme de chiffrement symétrique largement utilisé pour protéger les données sensibles.

Argon2
Algorithme de dérivation de clé (hashing) robuste, utilisé pour sécuriser les mots de passe.

Bcrypt
Algorithme de hachage de mots de passe résistant aux attaques par force brute.

CSRF (Cross-Site Request Forgery)
Attaque où un utilisateur authentifié est piégé pour exécuter une action non désirée sur une application web.

DAST (Dynamic Application Security Testing)
Analyse de sécurité réalisée sur une application en cours d'exécution pour détecter des vulnérabilités.

DevOps
Approche qui rapproche développement et opérations pour automatiser, collaborer et accélérer le cycle de livraison logicielle.

DevSecOps
Pratique qui intègre la sécurité tout au long du cycle de développement et de déploiement logiciel.

GitLeaks
Outil de détection de secrets (mots de passe, clés API...) dans les dépôts Git.

Heartbleed
Vulnérabilité critique découverte dans OpenSSL, ayant exposé massivement des données sensibles.

Injection SQL
Type d'attaque exploitant des failles de validation des entrées pour exécuter des requêtes malveillantes en base de données.

ISO / NIST
Organismes de normalisation qui définissent des standards de sécurité (ISO 27001, NIST Cybersecurity Framework).

JWT (JSON Web Token)
Format de jeton utilisé pour l'authentification et l'autorisation, nécessitant une gestion sécurisée.

Log4Shell
Vulnérabilité majeure découverte dans la librairie Log4j, permettant l'exécution de code à distance.

MFA (Multi-Factor Authentication)
Authentification forte nécessitant plusieurs facteurs (mot de passe + SMS, clé physique, etc.).

Observabilité
Capacité à collecter et analyser des logs, métriques et événements pour détecter des anomalies de sécurité.

OWASP Top 10
Référentiel reconnu listant les 10 principales vulnérabilités de sécurité des applications web.

RSA (Rivest–Shamir–Adleman)
Algorithme de chiffrement asymétrique utilisé pour l'échange sécurisé de données.

SAST (Static Application Security Testing)
Analyse statique du code source afin de détecter des failles de sécurité avant exécution.

SCA (Software Composition Analysis)
Analyse des librairies tierces et open source pour identifier des vulnérabilités connues.

SIEM (Security Information and Event Management)
Outil centralisant les logs et événements de sécurité pour corrélation et détection d'incidents.

SonarQube
Outil d'analyse de code qui détecte bugs, vulnérabilités et problèmes de qualité afin d'améliorer la fiabilité des applications.

SSRF (Server-Side Request Forgery)
Attaque où un serveur est manipulé pour effectuer des requêtes non prévues vers d'autres systèmes.

SSDF
Ensemble de bonnes pratiques défini par le NIST pour intégrer la sécurité à chaque étape du cycle de développement logiciel.

STRIDE
Méthodologie de modélisation des menaces (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege).

WAF (Web Application Firewall)
Pare-feu applicatif conçu pour protéger les applications web contre les attaques courantes.

Les auteurs de ce dossier tech

Philippe AUBERTIN

Passionné par les SI, le développement d'applications et l'optimisation des performances, j'ai créé AXOPEN en 2007. Au-delà des fonctions de dirigeant, je continue à développer des projets et à apporter mon expertise technique auprès de nos clients, mais aussi auprès des équipes en interne !

Thomas DA ROCHA

Lead développeur et membre du Génie Logiciel d'AXOPEN ! Grand fan de SpringBoot et Angular, j'ai également une appétence pour le devOps.

Arthur COMBE

Développeur par passion et de métier, membre du Génie Logiciel AXOPEN, mais surtout polyglotte ! Je parle couramment Java (Spring Boot Android) et JavaScript (NodeJS, AngularTS, ReactJS/Native, VueJS).

Autres leaders techniques contributeurs

Louis NOYARET

Nathan MITTELETTE

Grégory BOUC

AXOPEN, qui sommes-nous ?

Créée en 2007 à Lyon, notre entreprise AXOPEN est spécialisée dans les développements de projets informatiques métiers, techniques et complexes. Composée de plus de 60 collaborateurs (développeurs, chefs de projet et experts techniques) qui partagent des valeurs et des convictions communes autour de l'informatique, c'est une équipe 100% technique qui vous accompagne dans vos projets de développement !
Avec pour moteur la performance, la qualité et la durabilité du SI, nos équipes travaillent en collaboration avec des PME, ETI et grands comptes de tous secteurs d'activités pour répondre aux enjeux métiers spécifiques.

« Nous pensons que l'informatique ne doit pas être un sujet tabou où chacun conserve son savoir. Pour nous le partage est bénéfique et permet l'évolution positive des pratiques »