LES DÉFIS DE LA SÉCURITÉ DES APPLICATIONS : DES SOLUTIONS

L'importance de la résolution des problèmes liés aux applications Web pendant tout le cycle de vie du développement logiciel (SDLC): une analyse approfondie

---

KENNETH GRAF

Un livre blanc d'IBM

TABLE DES MATIÈRES

1. COMPRENDRE LES DÉFIS LIÉS À LA SÉCURITÉ DES APPLICATIONS

Aujourd'hui, les pirates Web qui attaquent vos applications sont capables de les utiliser contre vous pour vous rendre vulnérable, vous mettre dans l'embarras ou vous voler.
Les pare-feux et SSL sont devenus la norme et pourtant, d'après des enquêtes récentes, trois sites Web sur quatre sont vulnérables aux agressions, et la vaste majorité de ces offensives vise la sécurité des applications[1]. Les entreprises font confiance à la sécurité du réseau et de l'hôte, mais souvent, ces mesures ne suffisent pas pour empêcher ces attaques contre les applications Web.

La sécurité des applications diffère de celle du réseau et du système hôte. Les approches traditionnelles d'implémentation de la sécurité du réseau et de l'hôte ne conviennent pas à ce niveau. Dans ce livre blanc, nous vous expliquons pourquoi, nous vous recommandons les mesures à prendre et nous vous proposons une feuille de route pour renforcer la sécurité de vos applications.

1.1 LES DÉFIS CLASSIQUES DE LA SÉCURITÉ DES APPLICATIONS

Les entreprises vont devoir affronter une grande diversité de défis. Les tableaux suivants récapitulent les enjeux métier et techniques soulevés par certains d'entre eux, et pour lesquels nous vous présentons des solutions possibles dans la suite de ce livre blanc. La stratégie de Watchfire, basée sur une sensibilisation à la sécurité, une évaluation des risques, une tolérance zéro et des tests complets, peut aussi vous aider à résoudre vos problèmes de sécurité des applications.

1.1.1 Assurances : exemple d'une très grande société d'assurances gérant plusieurs millions de clients particuliers ou groupes de clients

Défis métier	• Mise en conformité avec les réglementations officielles. • Sécurisation proactive des enregistrements clients sensibles. • Intégration de la sécurité des applications à une stratégie de sécurité multi-niveaux, avec la prise en charge de plus de 4 000 praticiens dans 60 hôpitaux. • Suppression des coûts liés à la détection et à la correction des problèmes de sécurité au stade de post-production.
Défis techniques	• Croissance rapide du site. • Identification de nombreux défauts de sécurité des applications sur les sites (suite à un audit). • Garantie de la sécurisation des informations et des transactions des clients. • 95 % des données sont considérées comme confidentielles.

1.1.2 Finance : une grande banque commerciale américaine dotée d'un portefeuille de plus de $200 milliards

[1] "All-Out blitz against Web app attacks," *Network World*, 17 mai 2004. (http://www.networkworld.com/techinsider/2004/0517techinsidermain.html)

Défis métier	• Respect de la politique de l'entreprise exigeant l'intégration de la sécurité des applications au cycle de vie du développement. • Plus de 3 000 applications nouvelles et existantes. • Réduction du coût global du cycle de vie du développement. • La banque dépense >$1M par an en « piratage éthique » afin de détecter les vulnérabilités avant ou après un déploiement.
Défis techniques	• Site très connu et massivement visité. • Dispersion du développement d'applications dans de nombreuses unités commerciales. • Incohérence des tests manuels et de l'analyse du code par les développeurs. • Les développeurs n'ont ni les outils ni les connaissances des techniques de test de la sécurité.

1.1.3 Industrie pharmaceutique : un grand laboratoire de produits pharmaceutiques investissant beaucoup dans la recherche

Défis métier	• Mise en conformité avec les réglementations internes et officielles en matière de protection des données. • Évaluation exacte des risques associés à chaque application. • Volonté de restriction des ressources consacrées à la sécurisation des applications et des données.
Défis techniques	• Grand nombre de groupes de développement et de secteurs d'activités disparates. • Nombreuses acquisitions et partenariats stratégiques. • Données confidentielles soumises des réglementations très strictes. • Respect des réglementations internationales.

1.1.4 Loisirs et médias : une grande chaîne de télévision américaine

Défis métier	• Très forte exposition de la marque. • Profil haut de gamme très diversifié et caractéristiques média prêtant souvent à controverse.
Défis techniques	• Sites très dynamiques et très souvent modifiés. • Développement d'application entièrement décentralisé. • Petite équipe chargée de la sécurité des applications. • Plannings de production soumis à des délais très tendus.

1.2 DÉFINITION DE LA SÉCURITÉ DES APPLICATIONS

Le modèle de référence OSI (Open System Interconnection) définit sept couches de protocoles réseau, chaque message transitant par chacune de ces sept couches[2]. La couche supérieure, la

[2] International Organization for Standardization (www.iso.org)

couche 7, est la couche d'application qui utilise des protocoles tels que HTTP. HTTP permet de transmettre des messages incluant du contenu de type HTML, XML, SOAP et des services Web. Dans le cadre de ce livre blanc, nous allons étudier en particulier les offensives contre les applications véhiculées par HTTP.

Les pare-feux classiques peuvent se révéler inefficaces contre les offensives transitant par HTTP. L'agresseur de l'application utilise des requêtes HTTP valides en passant par des ports connus, de sorte que les pare-feux réseau, de par leur conception, autorisent volontairement ce trafic pourtant nuisible, tout simplement parce que la couche réseau considère qu'il s'agit d'un *bon* trafic. En effet, l'élément *nuisible* n'est pas la requête HTTP elle-même mais les données qu'elle contient. Souvent, ces données dangereuses sont des données entrées par l'utilisateur, spécialement formatées ou organisées dans le but de modifier le comportement de votre application. Les offensives contre les applications peuvent par exemple autoriser un accès sans restriction aux bases de données, exécuter des commandes système arbitraires ou modifier le contenu d'un site Web.

Une sécurité des applications bien pensée interdit à l'utilisateur mal intentionné de modifier le comportement de votre application.

1.2.1 Les facteurs les plus fréquents responsables de la sécurité insuffisante des applications

- Les obligations en matière de sécurité des applications, pour peu qu'elles aient été définies, sont souvent perçues comme impossibles à mettre en pratique, bien elles sont dotées d'une connotation négative (interdictions), ou encore elles s'avèrent trop floues.
- Le test de la sécurité des applications est effectué uniquement en cas d'audit.
- Les équipes chargées de la conception et de la définition des exigences des applications considèrent que la sécurité concerne l'équipe réseau ou informatique.
- Les procédures de test standard concernent principalement le comportement fonctionnel.
- Seuls les quelques rares « spécialistes de la sécurité » de l'entreprise sont conscients des menaces pesant sur la sécurité des applications.
- Le test de la sécurité des applications se limite à de courts créneaux au cours desquels des pirates dits éthiques tentent de simuler ce que feraient les vrais pirates.

2. LES MENACES CONTRE LA SÉCURITÉ DES APPLICATIONS

Les menaces contre la sécurité permettent de définir les technologies de sécurité qui peuvent être les plus efficaces pour défendre votre application. Il est souvent préférable d'envisager des mesures défensives d'ordre général avant d'opter pour une technologie particulière. En choisissant cette approche, vous aurez la certitude de choisir les meilleures technologies pour leurs mérites et non sur leur seule réputation.

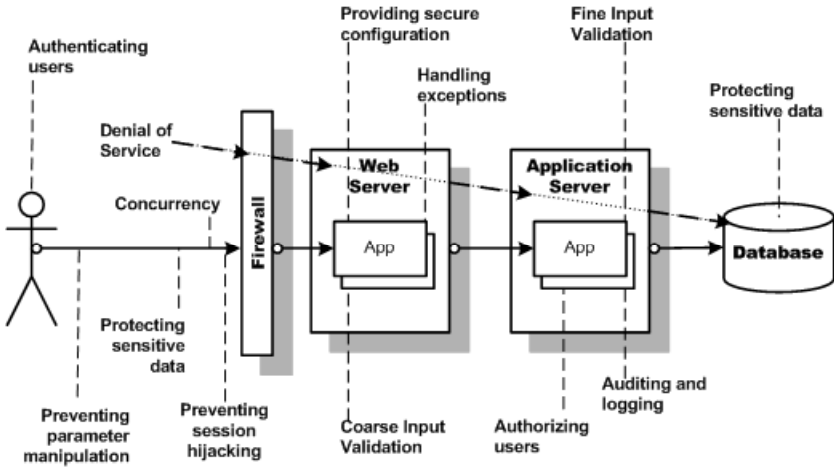


Figure 1 : Sécurité des applications Web : les enjeux les plus courants

Légendes du graphique

Authenticating users = Authentification des utilisateurs
Denial of service = Déni de service
Handling exceptions = Gestion des exceptions
Providing secure configuration = Configuration sécurisée
Fine Input Validation = Validation des entrées correctes
Protecting sensitive data = Protection des données sensibles
Concurrency = Simultanéité
Firewall = Pare-feu
Web server = Serveur Web
Application Server = Serveur d'application
Database = Base de données
App = App.
Protecting sensitive data = Protection des données sensibles
Preventing parameter manipulation = Prévention contre la manipulation de paramètres
Preventing session hijacking = Prévention du piratage de session
Coarse input Validation = Validation de données incorrectes
Authorizing users = Autorisation des utilisateurs
Auditing and logging = Audit et journalisation

Nous avons établi ci-après la liste des menaces les plus courantes et des solutions possibles. Le type de menaces diffère selon votre application.

2.1 USURPATION D'IDENTITÉ

Chaque fois qu'un utilisateur sollicite un accès à des informations non publiques, l'entreprise doit pouvoir vérifier l'authenticité de l'identité qu'il revendique. En général, vous pouvez empêcher l'usurpation d'identité en appliquant une authentification rigoureuse. Vous pouvez aussi vous protéger en sécurisant les informations d'authentification.

Exemples	• L'agresseur entre les informations d'authentification d'un autre utilisateur. • L'agresseur modifie le contenu d'un cookie ou d'un paramètre afin de se faire passer pour un autre utilisateur ou de faire croire que le cookie provient d'un autre serveur.

Erreurs courantes	• Utilisation d'une authentification basée sur les communications pour autoriser l'accès aux données d'un utilisateur. • Utilisation d'informations d'authentification non chiffrées qui peuvent être interceptées et réutilisées par un espion. • Stockage d'informations d'authentification dans des cookies ou des paramètres. • Utilisation de méthodes d'authentification « bricolées maison » ou non testées. • Le logiciel client n'est pas autorisé à authentifier l'hôte lorsque cela est nécessaire. • Utilisation d'une authentification provenant d'un domaine sécurisé erroné.
Solutions possibles	• Structures de sécurisation fournies par le système d'exploitation (par exemple Kerberos). • Utilisation de jetons chiffrés, tels que des cookies de session. • Utilisation de signatures numériques.

Tableau 1 : Menaces liées à l'usurpation d'identité

2.2 FALSIFICATION

La falsification consiste à modifier ou supprimer une ressource sans autorisation.

Exemples	• Défiguration d'un site Web. • Modification des données pendant leur transit.
Erreurs courantes	• Autorisation accordée à des sources de données sans validation préalable. • Assainissement des données en entrée afin d'empêcher l'exécution de code indésirable. • Droits d'exécution accordés à des comptes dont les privilèges ont été augmentés. • Absence de chiffrement des données sensibles.
Solutions possibles	• Utilisation de la sécurité au niveau du système d'exploitation pour verrouiller les fichiers, les répertoires et les autres ressources. • Validation de vos données. • Affectation d'un code de hachage ou d'une signature aux données en transit (SSL ou IPsec).

Tableau 2 : Menaces liées à la falsification

2.3 RÉPUDIATION

La répudiation consiste à nier qu'une action s'est produite. Une offensive de répudiation a pour but de tenter de détruire, de masquer ou de modifier des preuves.

Exemples	Suppression de journaux. Usurpation d'identités pour faire une demande de modifications.
Erreurs courantes	Processus d'autorisation et d'authentification insuffisant, voire inexistant. Journalisation inadaptée. Autorisation de présence d'informations sensibles sur des canaux de communications non sécurisés.

Solutions possibles	Règles strictes en matière d'authentification, d'audits, d'enregistrements de transaction, de journaux ou de signatures numériques.

Tableau 3 : Menaces liées à la répudiation

2.4 DIVULGATION D'INFORMATIONS

La divulgation d'informations consiste simplement à révéler publiquement des informations privées. La gravité d'une telle attaque varie selon la quantité et le degré de sensibilité des informations diffusées. La falsification de données est la capacité à modifier des informations divulguées.

Exemples	• Vol de mots de passe. • Obtention d'informations de carte de crédit ou d'autres informations personnelles identifiables (PII) similaires. • Obtention d'informations sur le code source de l'application et/ou de ses systèmes hôte.
Erreurs courantes	• Possibilité pour un utilisateur authentifié d'accéder aux données d'autres utilisateurs. • Autorisation de présence d'informations sensibles sur des canaux de communications non sécurisés. • Choix inadapté des algorithmes et des clefs de chiffrement.
Solutions possibles	• Stockage non permanent des informations pendant une session. • Utilisation du hachage et du chiffrement chaque fois que possible. • Comparaison des données de l'utilisateur avec l'authentification de l'utilisateur.

Tableau 4 : Menaces liées à la divulgation de l'information

2.5 DÉNI DE SERVICE (DOS)

Une attaque de type Déni de service (DoS) a pour effet de réduire la disponibilité normale d'une application. Les attaques DoS revêtent deux formes : 1) l'invasion par surcharge, qui consiste à envoyer un nombre très élevé de messages pour faire tomber un serveur ; 2) le verrouillage, où la requête a pour effet de rendre les temps de réponse du serveur extrêmement longs en consommant des ressources ou en rendant les ressources indisponibles.

Ce type d'offensives peut se dérouler à n'importe quel niveau du modèle OSI. Elles sont relativement faciles à orchestrer, mais difficiles à contrer.

Exemples	• Envoi d'un trop grand nombre de requêtes simultanées à l'application. • Envoi de requêtes qui provoquent le redémarrage de l'application ou des temps de réponse très longs.
Erreurs courantes	• Placement d'un trop grand nombre d'applications ou d'applications conflictuelles sur un seul serveur. • Test d'unités incomplet.

Solutions possibles	• Filtrage des paquets avec un pare-feu. • Utilisation d'un programme d'équilibrage de charge pour réduire le nombre de requêtes émanant d'une seule source. • Utilisation de protocoles asynchrones pour gérer les requêtes demandant des calculs intensifs avec une reprise des erreurs adaptée.

Tableau 5 : Menaces liées au Déni de service (DoS)

2.6 Elévation DES PRIVILÈGES

Une augmentation des privilèges signifie que l'on reçoit les droits plus élevés que ceux qui sont normalement attribués.

Exemples	• Un utilisateur se voit affecter des droits d'administration. • Un employé accède à un rôle de manager.
Erreurs courantes	• Exécution de processus du serveur Web avec les droits « root » ou « administrateur ». • Erreurs de codage permettant des dépassements de la mémoire tampon, plaçant l'application dans un état de déboguage élevé.
Solutions possibles	• Utilisation d'un contexte de privilèges réduits au minimum chaque fois que possible. • Utilisation de langages sécurisés et d'options de compilateur pour éviter ou contrôler les dépassements du tampon.

Tableau 6 : Menaces liées à l' élévation de privilèges

3. ANALYSE DES PROBLÈMES DE SÉCURITÉ DES APPLICATIONS EXISTANTES

Le cycle de vie des erreurs liées à la sécurité des applications (apparition, détection, correction) est exactement identique à celui de n'importe quel autre type d'erreur liée aux applications. Ce livre blanc a pour principal thème l'amélioration des processus spécifiques à la sécurité utilisés lors de la création d'applications. Les améliorations générales des processus relevant du cycle de vie du développement logiciel (cycle SDLC) n'entrent pas dans le cadre de cet article.

Nous vous conseillons de consulter en premier l'Annexe D : « Utilisation de CMMI pour améliorer la sécurité des applications » si vous avez besoin d'apporter des améliorations globales de processus en plus de la sécurité.

Quasiment toutes les entreprises possèdent des applications et des systèmes anciens déjà déployés qu'elles ont besoin de protéger. Mais le travail d'évaluation et de gestion de l'infrastructure est exigeant, coûteux et doit s'effectuer de manière continue. Il existe de nombreux ouvrages et sources de référence sur le Web qui vous expliquent comment gérer de façon optimale la sécurité d'une application déjà en place. En fait, la plupart de ces références

partent du principe que vous n'avez pas la possibilité d'apporter des corrections à de telles applications. Dans ce livre blanc, même si les approches à la sécurité applicative que nous vous proposons traitent des avantages qu'elles apportent aux applications existantes, leur utilisation dans le contexte de développement des nouvelles applications ou dont la conception a été revue peut être encore plus intéressante.

Nous vous recommandons de prendre dès maintenant les mesures suivantes pour vos applications déjà existantes :

3.1 DECOUVERTE ET INVENTAIRE

- Effectuez un inventaire complet de tous vos systèmes et applications. Recueillez des informations techniques (IP, DNS, système d'exploitation utilisé, etc.) ainsi que des informations métier, par exemple identification de la personne ayant autorisé le déploiement ou de la personne à prévenir en cas d'arrêt de l'application.

- Recherchez les vulnérabilités et les failles de systèmes. Renseignez-vous notamment sur les offensives connues susceptibles de menacer votre système d'exploitation, votre serveur Web et les autres produits tiers dont vous dépendez. Ces offensives sont normalement publiées et sont immédiatement disponibles. Dans l'idéal, avant de deployer votre application sur un serveur, appliquez-lui des correctifs, renforcez-le et analysez-le.

- Recherchez les vulnérabilités de vos applications aux offensives connues. Lors d'une telle évaluation, examinez notamment les requêtes HTTP utilisées par votre application et essayez de manipuler les données. Pour ce type d'évaluation, un test de type « boîte noire » est généralement utilisé.

- Testez la gestion de l'authentification des applications et des droits des utilisateurs.

- Arrêtez tous les services inconnus.

3.2 EVALUATION ET ANALYSE DE LA RÉPARTITION DES RISQUES

- Évaluez les risques auxquels sont exposés vos applications et vos systèmes. Prêtez une attention particulière aux magasins de données, au contrôle des accès, au profilage et aux droits des utilisateurs.

- Établissez un classement par priorités des vulnérabilités des applications détectées lors des évaluations. Vous pouvez notamment vous aider de l'Annexe E : Classification des risques avec le modèle Microsoft DREAD.

- Évaluez votre conformité aux réglementations internes, sectorielles et officielles. Définissez ce qui est acceptable et ce qui ne l'est pas.

3.3 MESURES DE PROTECTION ET CONTRÔLE DES DOMMAGES

- Appliquez des correctifs, si vous en disposez, à l'application et/ou à l'infrastructure.

- Il ne sera pas toujours possible de remédier aux problèmes de sécurité d'une application. Dans ce cas, vous devez protéger au maximum le défaut de sécurité afin d'empêcher ou de réduire son exposition aux agressions. Vous pouvez mettre en place un pare-feu d'application pour la protéger, ou restreindre, désactiver ou relocaliser l'application.

3.4 CONTINUITÉ DE LA SURVEILLANCE ET DE L'ANALYSE

- Les évaluations sont planifiées dans le cadre de processus documentés de gestion des modifications. Cette opération conclut le cycle en lançant une nouvelle phase de reconnaissance.

4. RÉSOLUTION DES ERREURS PENDANT TOUT LE CYCLE DE VIE DU DÉVELOPPEMENT LOGICIEL

4.1 COÛT DE LA CORRECTION DES ERREURS DANS LE CYCLE DE VIE DU DÉVELOPPEMENT LOGICIEL

Toutes les applications sont soumises à des pressions au cours du cycle de vie du développement logiciel (SDLC), telles que les délais de mise au marché à respecter pour rester concurrentiel, une complexité croissante, une augmentation des risques métier. Plus les erreurs de l'application passent inaperçues pendant longtemps, plus les coûts augmentent, et parfois en flèche. Le Tableau 7 est extrait d'une enquête NIST de 2002 (lien direct : http://www.nist.gov/director/prog-ofc/report02-3.pdf)

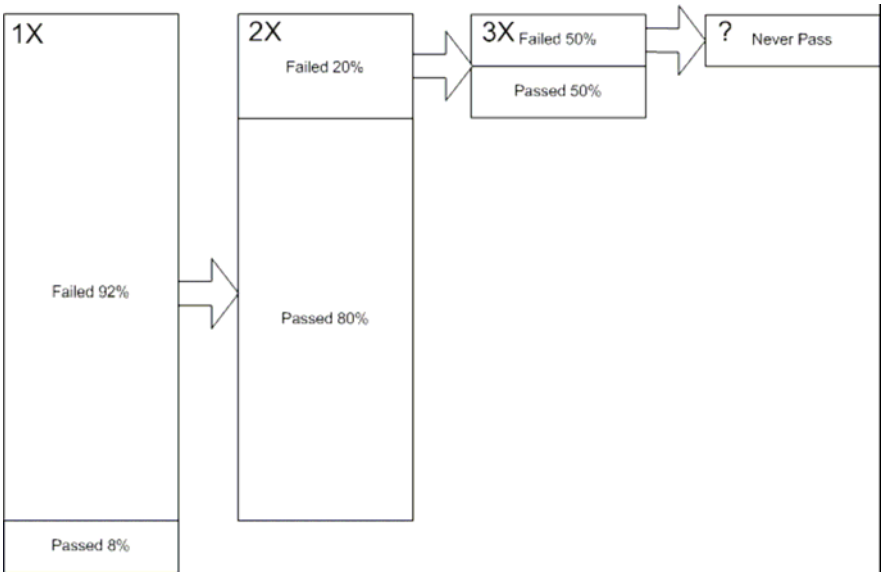
	Erreur détectée à la conception	Erreur détectée au codage	Erreur détectée à l'intégration	Erreur détectée dans la version bêta	Erreur détectée dans la version finale approuvée
Erreurs de conception	**1x**	**5x**	**10x**	**15x**	**30x**
Erreurs de codage		**1x**	**10x**	**20x**	**30x**
Erreurs d'intégration			**1x**	**10x**	**20x**

Tableau 7 : Coûts relatifs calculés en fonction du délai écoulé entre l'apparition de l'erreur et sa détection

Lorsqu'une erreur de conception est détectée lors de la version finale approuvée de l'application, le coût nécessaire pour la corriger est 30 fois supérieur à ce qu'il aurait été si l'erreur avait été corrigée lors de la phase de conception. Et encore, ce coût ne correspond qu'au coût de la correction par l'équipe responsable de l'application : il n'intègre pas les autres facteurs tels que la perte d'une part de marché, le ternissement de la réputation de l'entreprise ou le degré de satisfaction des clients.

Une enquête réalisée par Sanctum (racheté par Watchfire en 2004) et portant sur plus de 100 applications utilisées par les sites de grandes entreprises et d'organismes publics révèle quelques chiffres très dérangeants sur le taux de non-conformité à la sécurité. L'enquête a ainsi

constaté que 92 % de toutes ces applications échouaient aux tests de sécurité effectués lors des phases d'intégration ou de production. La durée moyenne de résolution des erreurs était de 2 mois et demi, et le coût pour l'équipe responsable avoisinait les $25 millions. Lors d'un nouveau test des applications défaillantes, 20 % des tests de sécurité (soit 16 % du total) se sont soldés par un nouvel échec. La moitié des applications ayant essuyé un second échec (8 % au total) n'ont jamais réussi ces tests.



Légende du graphique

Failed 20% = Échec 20 %
Failed 50% = Échec 50 %
Passed 50% = Réussite 50 %
Never Pass = Échec définitif
Failed 92% = Échec 92 %
Passed 8% = Réussite 8 %
Passed 80% = Réussite 80 %

Compte tenu de la forte probabilité d'échec aux tests de sécurité du coût représenté par une détection tardive d'erreurs lors du cycle de vie du développement logiciel, il est donc vivement conseillé d'améliorer les tests de sécurité pendant tout ce cycle et de dépister très tôt les erreurs de sécurité.

4.2 TYPES D'ERREURS RENCONTRES DANS LE CYCLE DE VIE DU DÉVELOPPEMENT LOGICIEL

Avant d'aborder les moyens de détection et de correction des erreurs, nous allons passer en revue les différents types d'erreurs que nous tentons de corriger, en fonction du moment auquel elles apparaissent dans le cycle de vie du développement logiciel.

4.2.1 Erreurs au stade de la définition des exigences

L'équipe responsable des exigences dispose en général d'un tableau global des critères de sécurité que son application va devoir respecter. Mais si cette équipe ne connaît pas les menaces pesant sur la sécurité des applications, il y a peu de chances qu'elle communique les exigences spécifiques à l'application aux concepteurs, notamment la nécessité de rechercher les menaces ciblant cette application en particulier.

4.2.2 Erreurs au stade de la conception

Si vous avez défini un ensemble d'exigences approprié, à quel moment votre équipe de conception peut-elle faire fausse route ? Cette équipe doit-elle aussi être sensibilisée aux menaces liées à la sécurité de l'application ? Un choix technologique insatisfaisant ou mal adapté peut laisser penser aux concepteurs, à tort, qu'ils ont pris les bonnes mesures pour répondre à une exigence de sécurité particulière. Or, cette connaissance insuffisante de la sécurité signifie que le système de test va être incomplet ou inefficace, voire inexistant.

4.2.3 Erreurs au stade du codage

Vous êtes doté d'une conception saine à partir de laquelle vos développeurs vont pouvoir travailler. Quelles erreurs risquent-ils d'introduire ? En général, l'erreur se produit si, au lieu de rédiger du code entièrement nouveau, ils réutilisent du code comportant des imperfections, ou génèrent du code à l'aide d'un assistant IDE non sécurisé. Le danger est d'effectuer une validation des données incorrecte ou de ne pas utiliser correctement les fonctions de sécurité du système choisi pour l'application.

4.2.3 Erreurs se produisant à un stade ultérieur

Dans la plupart des entreprises, les connaissances en matière de sécurité des applications sont souvent détenues par une poignée de collaborateurs membres d'équipes d'audit de sécurité chargées de réaliser des intrusions ou des piratages volontaires pour tester les défenses de l'entreprise. Ces équipes opèrent dans des créneaux limités et examinent l'application vers la fin du cycle de vie du développement logiciel, dans l'espoir d'intercepter des erreurs de sécurité avant la commercialisation de l'application.

La centralisation de cette fonction est due au fait qu'un pirate éthique compétent est rare (comprendre *cher*), et que ces équipes réalisent souvent des audits commandés de systèmes de production. Comme on le verra, une telle approche constitue un goulot d'étranglement, est coûteuse et ne permet pas de détecter toute la gamme d'erreurs de sécurité des applications.

4.3 APPROCHES GÉNÉRALES DU TEST DE LA SÉCURITÉ DES APPLICATIONS

Les erreurs de sécurité des applications ne sont que des erreurs. Même s'il est nécessaire de mettre en œuvre des compétences spécifiques pour créer les tests de détection des erreurs de sécurité, lorsque ce type d'erreur est décelé, le processus de correction est entièrement identique à celui employé pour remédier aux autres erreurs.

4.3.1 Tests manuels

Les tests d'infiltration ou de sécurité sont souvent effectués par une petite équipe de spécialistes. Ils font souvent appel à des outils et des scripts connus.

Avantages	• Génération de tests bien ciblés adaptés à des fonctions d'application spécifiques.

Inconvénients	• Les tests de sécurité sont effectués par un nombre restreint de spécialistes. • Risque d'erreurs humaines. • Poste de dépense à récurrence très fréquente. • Les contraintes de temps limitent les applications couvertes.

4.3.2 Tests automatiques

Les tests automatiques sont en général conçus de l'une des deux façons suivantes : 1) une approche de bas en haut, le développeur de code créant spécialement des tests destinés à des fonctions ou des méthodes données ; 2) une approche de haut en bas, les tests étant conçus par les équipes d'assurance qualité du point de vue de l'utilisateur final.

La création et la maintenance des tests automatiques nécessitent des investissements plus élevés que les tests manuels. Ces frais sont normalement compensés par des améliorations au niveau de la qualité, la diminution des tests de réception, et l'amélioration des processus de développement itératif.

4.3.3 Le test de la boîte noire

Le test de la boîte noire, parfois appelé test système, est une approche de haut en bas. Il part du principe que vous ignorez les rouages internes de l'application. Votre connaissance de l'application se limite à en examiner les entrées et les sorties. Il s'agit de la forme la plus courante des tests de sécurité, utilisée par les responsables d'audit, les testeurs manuels et les pirates éthiques. Le ou les tests consistent à modifier des entrées utilisateur « normales » pour amener l'application à se comporter de façon inattendue.

Avantages	• Test ne nécessitant que peu de connaissances, voire aucune, de l'application. • Outils bien rodés d'automatisation des tests.
Inconvénients	• Le test peut uniquement être exécuté lorsque toutes les composantes de l'application sont prêtes à être testées (en général dans un environnement avancé de transfert ou de production). • Des modifications des entrées utilisateur peuvent provoquer un grand nombre de transactions. Ne pas tenir compte des résultats de ces transactions ou parvenir à les annuler est souvent problématique pour les systèmes de production. • En raison de la visibilité limitée de l'application, des défauts peuvent passer inaperçus.

4.3.4 Le test de la boîte blanche

Le test de la boîte blanche, parfois appelé « test du source », teste chacun des composants de votre application. Souvent, ce test est effectué au niveau d'une méthode ou d'une fonction. Il est exécuté de façon à déceler des erreurs de fonctions et est souvent associé à des outils d'analyse de code et soumis à des examens par des homologues.

Avantages	• Reconnaissance bien définie des défauts des fonctions testées.

	• Intégrations bien rodées aux environnements IDE des développeurs.
Inconvénients	• Ce type de test, portant sur le source, ne reconnaît pas les défauts liés aux exigences et à la conception. • Reconnaissance médiocre des erreurs de sécurité, car de nombreuses offensives portent sur plusieurs composants ou ont une durée particulière non prise en charge par le test unitaire. • Les tests sont souvent écrits par l'auteur du code. Or, si le développeur n'est pas formé à la sécurité, il ne peut savoir de quels tests il a besoin.

4.3.5 Le test de la boîte grise (utilisation d'un système de test défini par l'application)

Ce système de test associe à la fois des tests de type boîte noire et boîte blanche. Généralement, on utilise un tel système pour créer l'état de l'application et le test d'événement qui ne sont pas disponibles dans les outils de test disponibles dans le commerce. L'Annexe F: « Test de sécurité déterminé par les événements », donne un exemple possible d'un tel système.

Avantages	• C'est la méthode la plus complète, associant à la fois test au niveau du système et de l'unité. • Les tests déterminés par les actions fournissent des tests basés sur l'état et le moment d'occurrence. • Les agents et/ou les proxys peuvent être utilisés pour des tests basés sur les causes et les effets. • Le système de test peut être créé pour permettre des tests d'audit en production sans impact sur les données de production. • Le système de test peut surveiller les flux de données via l'application.
Inconvénients	• La méthode doit être définie dans le cadre des phases de définition des exigences et de conception. • Le travail nécessaire à la création d'un système de test est souvent aussi important que l'application à tester.

5. DÉFINITION D'UNE STRATÉGIE

Watchfire conseille aux entreprises de travailler sur quatre catégories : 1) Sensibilisation à la sécurité ; 2) Classification des risques et responsabilités des applications ; 3) Application de la tolérance zéro et 4) Intégration des tests de la sécurité au processus de développement. Bien qu'il soit possible d'obtenir des améliorations en se concentrant sur une ou deux de ces catégories de tâches, vous obtiendrez des résultats optimaux en exécutant toutes les catégories.

5.1 SENSIBILISATION A LA SÉCURITÉ

Cette catégorie comprend des tâches de formation, de communication et de surveillance. Pour les mener à bien de manière correcte, nous conseillons aux entreprises de mettre en place une approche consultative ou collaborative. Les modifications unilatérales, les rapports que personne ne lit et les règles que personne ne respecte sont bien évidemment contre-productifs.

Nos recommandations ci-après ne sont que des suggestions qui doivent être adaptées en fonction des besoins de chacun.

Formation	• Organisation d'une demi-journée annuelle de formation à la sécurité des applications à l'attention de tous les membres de l'équipe d'application, notamment les développeurs, les spécialistes d'assurance qualité, les analystes et les managers. Cette formation doit aborder les offensives existantes, la façon dont elles sont créées, et le processus de résolution recommandé. Elle doit fournir des informations sur la stratégie actuelle de sécurité de l'entreprise et présenter ses meilleures pratiques de sécurité. • Chaque développeur doit suivre une formation spécifique au système de sécurité choisi. La durée standard de la formation (de 1 à 5 jours) varie selon le système et peut parfois prendre la forme d'une auto-formation. Tout système de sécurité digne de ce nom doit comporter des fonctions de sécurité pré-intégrées qui doivent être maîtrisées. • Outils de sécurité du commerce : suivre la formation recommandée par les éditeurs de ces outils.
Communication	• Les directives en matière de meilleures pratiques de sécurité doivent être rédigées à partir de l'expérience de toutes les équipes et de tous les secteurs d'activités. Ce document doit être court (<10 pages), aller droit au but et pouvoir s'appliquer à toute l'entreprise. • Développement de processus comprenant une surveillance par des homologues. • L'équipe de sécurité doit définir une personne ou une méthode de contact avec chaque équipe d'application pour pouvoir l'aider en cas de problèmes liés aux exigences et à la conception des applications.
Surveillance	• La stratégie de sécurité de chacune des applications en production doit être connue à tout moment. • Les erreurs de sécurité doivent être suivies par le biais d'infrastructures normales de suivi et de reporting des défauts afin de veiller à ce que toutes les parties prenantes disposent d'une visibilité appropriée.

Tableau 8 : Tâches de sensibilisation à la sécurité

5.2 CLASSIFICATION DES RISQUES ET DES RESPONSABILITÉS LIÉS AUX APPLICATIONS

Toutes les entreprises ont des problèmes de ressources limitées. Elles doivent gérer les priorités et la sécurité n'y fait pas exception. Nous sommes aperçus que les entreprises ont souvent du mal à budgétiser en valeur réelle les risques liés aux applications, et que même lorsqu'elles le font correctement, leur solution n'est guère différente des modèles plus modérés tels que DREAD (voir l'Annexe E). A moins de disposer de ressources illimitées, les tâches suivantes sont indispensables :

- Définition des seuils de risque. La date à laquelle l'équipe de sécurité doit mettre fin aux services d'application doit être prise en compte.

- Classification des applications en fonction de leurs facteurs de risque (par exemple utilisateurs internes/externes ou déploiement réseau, Internet, Intranet/Extranet).

- Toutes les analyses de sécurité d'une application doivent être synthétisées dans un rapport de risques qui est comparé avec les seuils de risques définis.

5.3 APPLICATION D'UNE TOLÉRANCE ZÉRO

Cette catégorie peut sembler de prime abord complexe. Elle pourrait aussi s'intituler « Nul n'est censé ignorer la loi ». Si votre entreprise a établi des règles de sécurité bien définies, vous devez savoir avant le déploiement si votre application y est conforme ou non.

- L'équipe de développement de l'application doit savoir dès le début du projet quels tests elle devra réussir.
- Les exigences et la conception de l'application doivent être étudiés de façon formelle afin de déceler les problèmes de sécurité avant le début du codage.
- S'il existe une raison majeure pour qu'une application déroge aux règles de sécurité d'une entreprise, la dérogation doit être accordée par le directeur de l'informatique dans le cadre d'un processus d'approbation de la conception.
- Des règles de sécurité claires doivent être en place.
- L'objectif de taux d'erreurs maximum de l'équipe d'application, toutes erreurs confondues, doit être inférieur à une seule erreur pour 10 000 lignes de code.

5.4 INTÉGRATION DES TESTS DE SÉCURITÉ AU PROCESSUS DE DÉVELOPPEMENT

Ce groupe de tâches est présenté en dernier pour une raison spécifique. Pour être efficace, il dépend de la bonne progression des autres groupes de tâches. Une fois sa mission achevée, c'est aussi ce groupe qui offre le plus important retour sur investissements. C'est le seul qui a un impact significatif sur la conception, le développement et le test de votre application.

- Les tests de sécurité imposés à une application doivent correspondre à des exigences fonctionnelles explicites.
- Le système de tests doit pouvoir être exécuté à la demande.
- Le système de tests de l'application doit comprendre des tests d'unité et de système, ainsi que tous les tests requis pour remédier aux menaces pesant sur l'application.
- Le système de tests doit être automatisé.
- La conception du système de tests doit permettre un test d'audit en production.
- Lors de la création du système de tests de l'application, veillez à inclure des tests basés sur un modèle déterminé par les événements comme celui défini à l'Annexe F : « Test de sécurité déterminé par les événements ».
- Bien qu'ils ne soient pas obligatoires, les processus de développement souples tels que XP et SCRUM sont les méthodologies de développement de sécurité conseillées.
- La conception de l'application doit permettre l'exécution du système de tests pendant les phases de codage, de test, d'intégration et de production.

6. VALIDATION DE VOTRE MÉTHODOLOGIE

Il est important de bien progresser dans les quatre domaines que nous venons de décrire.

6.1 SENSIBILISATION A LA SÉCURITÉ

Toute l'équipe de développement doit savoir sur le plan conceptuel ce que sont les offensives contre les applications et quel est leur mode de fonctionnement.

Principales mesures à prendre :
• Formation annuelle à la sécurité des applications.
• Cette formation peut être associée à une formation organisationnelle spéciale aux règles.

6.2 CLASSIFICATION DES RISQUES ET DES RESPONSABILITÉS LIÉS AUX APPLICATIONS

Le livrable de ce domaine consiste en une base de données classifiant toutes les applications en fonction de leurs risques. Chaque équipe doit connaître l'état de son application et être capable de le comparer aux autres applications déjà déployées.

Principales mesures à prendre :
• Collecte des données sur les risques des applications.
• Comparaison de l'application par rapport au profil de sécurité établi par l'entreprise.
• Les exigences de sécurité de l'application doivent être définies en fonction des règles de référence.

6.3 APPLICATION D'UNE TOLÉRANCE ZÉRO

Lorsque l'équipe a été sensibilisée aux offensives contre la sécurité, et que les risques de l'application ont été évalués, l'équipe doit assumer sa responsabilité vis-à-vis d'une distribution sécurisée. Le secret pour obtenir un résultat concluant consiste à informer dès le départ l'équipe des tests auxquels va être soumise l'application.

Principales mesures à prendre :

- Les exceptions à la sécurité ne doivent être admises que pendant la phase de conception et uniquement avec l'approbation du responsable concerné.
- L'ensemble des tests de sécurité dot être présenté à l'équipe de l'application comme une exigence.

6.4 TEST DE LA SÉCURITÉ

Ce domaine rassemble les résultats des trois premiers domaines. L'équipe a été sensibilisée, les risques sont connus et une règle de tolérance zéro a été mise en place. Pour pouvoir assurer la qualité et la sécurité des applications de façon prévisible, des outils de test automatique doivent être déployés au cours de tout le cycle de vie du développement logiciel.

Principales mesures à prendre :

- Outils automatiques de test de la sécurité pour les développeurs, les équipes d'assurance qualité et les responsables d'audits.
- Possibilité d'exécuter des tests complets de la sécurité à n'importe quel moment du processus de développement.

ANNEXE A : PHASE DE DEFINITION DES EXIGENCES – CONSIDÉRATIONS SUR LA SÉCURITÉ

La liste suivante reprend un certain nombre d'aspects à prendre en compte lors de la phase de définition des exigences de l'application. Vous pouvez la compléter en y ajoutant des considérations qui vous sont propres.

Environnement de l'application

- Identification, compréhension et intégration des règles de sécurité de l'entreprise.
- Reconnaissance des restrictions d'infrastructure (restrictions liées aux services, aux protocoles et aux pare-feux).
- Identification des restrictions liées à l'hébergement hôte (sous-réseau, réseau VPN, exécution dans une zone protégée).
- Définition de la configuration de déploiement de l'application.
- Définition des structures du domaine réseau, de la mise en cluster et des serveurs d'applications à distance.
- Identification des serveurs de base de données.
- Bonne connaissance des fonctions de communication fournies par l'environnement.
- La conception doit satisfaire les considérations liées aux fermes Web, notamment la gestion de l'état de la session, les clés de chiffrement spécifiques à la machine, SSL (Secure Sockets Layer), les problèmes de déploiement de certificats et les profils itinérants.
- Si l'application utilise SSL, l'autorité de certification et les types de certificats doivent être identifiés.
- La conception doit satisfaire les critères d'évolutivité et de performances requis.
- Bonne connaissance du niveau de sécurisation du code.

Validation des entrées et des données

- Toute entrée est par définition suspecte.

Authentification

- Identification de toutes les limites de sécurisation.
- Identification de comptes et/ou de ressources enfreignant les limites de sécurisation.
- Utilisation de comptes dotés de privilèges minimum.
- Élaboration de règles de gestion des comptes.
- Si la stratégie de sécurité nécessite un mot de passe renforcé, mise en vigueur de cette obligation.
- Chiffrement de la communication des informations d'authentification des utilisateurs (SSL, VPN, IPsec).
- Utilisation systématique de connexions chiffrées pour la transmission des informations d'authentification (jetons, cookies, tickets, etc.).
- Renvoi seulement d'un minimum d'informations d'erreur en cas d'échec de l'authentification.

Gestion de session

- Chaque session doit avoir une durée limitée.
- L'état d'une session doit être protégé contre les accès non autorisés.
- Les ID de session ne doivent pas être transmis dans des chaînes de requête.

ANNEXE B : PHASE DE CONCEPTION – CONSIDÉRATIONS SUR LA SÉCURITÉ

La liste suivante reprend un certain nombre d'aspects à prendre en compte lors de la phase de conception de l'application. Vous pouvez la compléter en y ajoutant des considérations qui vous sont propres.

Validation des entrées et des données

- Toute entrée est par définition suspecte.
- La validation des entrées est effectuée sur un serveur contrôlé par l'application.
- La validation des entrées côté client peut être effectuée pour des raisons liées à l'interface graphique, mais ne se substitue pas à la validation côté serveur.
- La conception doit résoudre les problèmes potentiels liés à la mise au format canonique, l'injection de code SQL et le scriptage inter-sites.
- Tous les points d'entrée et les limites de frontières sont identifiés.

Authentification

- Séparation des domaines en accès public et à accès restreint.
- Identification des comptes et/ou ressources qui enfreignent les limites de sécurisation.
- Identification des comptes effectuant la maintenance ou l'administration de l'application.
- Stockage sécurisé des informations d'authentification envoyées par les utilisateurs.
- Chiffrement des informations d'authentification des utilisateurs (SSL, VPN, IPsec).
- La conception doit permettre d'identifier l'identité utilisée pour l'authentification auprès de la base de données.

Autorisation

- Toutes les identités qui sont utilisées par l'application sont identifiées et les ressources auxquelles chaque identité accède sont connues.
- La conception du rôle doit permettre une séparation suffisante des privilèges (c'est-à-dire prendre en compte la granularité des autorisations).
- La conception doit identifier les exigences de sécurité pour l'accès au code.
- Identification des ressources et des opérations privilégiées.

Gestion de la configuration

- Sécurisation des interfaces d'administration (utilisation d'une authentification et d'une autorisation renforcées).
- Sécurisation des canaux d'administration à distance.
- Séparation des privilèges d'administration en fonction des rôles (par exemple, développeur de contenu de site ou administrateur système).
- Utilisation de comptes dotés de privilèges minimum et de comptes de maintenance.

Données sensibles

- Les données secrètes ne sont pas stockées sauf si cela est indispensable. D'autres méthodes doivent être évaluées lors de la phase de conception.
- Identification d'algorithmes de chiffrement et de tailles de clé pour le stockage sécurisé des secrets.
- La conception doit identifier des mécanismes de protection pour les données sensibles envoyées via le réseau.

Gestion de session

- Utilisation de SSL pour protéger les cookies d'authentification.
- Chiffrement du contenu des cookies d'authentification.

Cryptographie

- Sécurisation des clés de chiffrement.
- Utilisation uniquement de bibliothèques et de services de chiffrement connus.
- Identification des algorithmes de chiffrement et des tailles de clé appropriés.
- Identification de la méthodologie de sécurisation des clés de chiffrement.

Gestion des exceptions

- Définition d'une approche standard de la gestion de la structure des exceptions.
- Identification dès la conception des messages d'erreur génériques renvoyés au client.

Audit et journalisation

- Identification du niveau d'audit et de journalisation nécessaire à l'application.
- Identification des principaux paramètres à journaliser et auditer.
- Identification du stockage, de la sécurité et de l'analyse des fichiers journaux d'application.
- La conception doit tenir compte de la façon dont l'identité de l'émetteur peut être transmise à plusieurs niveaux (niveau du système d'exploitation ou de l'application) à des fins d'audit.

ANNEXE C : PHASE DE CODAGE – CONSIDÉRATIONS SUR LA SÉCURITÉ

La liste suivante reprend un certain nombre d'aspects à prendre en compte lors de la phase de codage de l'application. Vous pouvez la compléter en y ajoutant des considérations qui vous sont propres.

Pratiques de codage

- Mise en place d'une utilisation correcte de l'authentification et de l'autorisation.
- Si votre application nécessite des fonctions qui vous contraignent à alléger ou modifier les paramètres de sécurité par défaut, testez-en les effets et évaluez avec exactitude les conséquences avant toute modification.
- N'intégrez pas de données secrètes dans le code. En matière de sécurité, le choix de l'opacité n'est pas jouable.
- Si vous n'êtes pas propriétaire d'un élément donné, ne le sécurisez pas.
- N'exposez pas les informations qui ne sont pas nécessaires.
- Mettez en place une gestion des erreurs ne provoquant pas d'arrêt total.
- En cas de basculement vers un mode sûr : n'affichez pas les traces de pile et ne laissez pas les données sensibles sans protection.

Validation des entrées et des données

- Toute entrée est par définition suspecte.
- Validation de tous les paramètres d'entrée (y compris les zones d'entrée, les chaînes de requête, les cookies et les en-têtes HTTP).
- La validation des entrées côté client peut être effectuée pour des raisons liées à l'interface graphique, mais ne se substitue pas à la validation côté serveur.
- Le modèle conseillé est celui de la validation positive (acceptation uniquement des entrées connues et correctes) et non celui de la validation négative (rejet des entrées connues incorrectes).
- Validation des données par type, longueur, format et plage.
- Encodage des sorties contenant des entrées en HTML ou URL.

Authentification

- Stockage des mots de passe avec processus d'authentification Digest et clé Salt.
- Renvoi seulement d'un minimum d'informations d'erreur en cas d'échec de l'authentification.
- La prise de décisions affectant la sécurité ne doit pas s'appuyer uniquement sur les informations de l'en-tête HTTP.

Autorisation

- La connexion à la base de données de l'application permet uniquement d'accéder à des procédures mémorisées à accès spécifique et ne permet pas l'accès direct aux tables.
- Restriction de l'accès aux ressources au niveau système.

Gestion de la configuration

- Sécurisation des magasins de configuration.
- Les données secrètes de configuration ne doivent pas figurer en texte clair dans les fichiers de configuration.
- Utilisation de comptes dotés de privilèges minimum et de comptes de maintenance.

Données sensibles

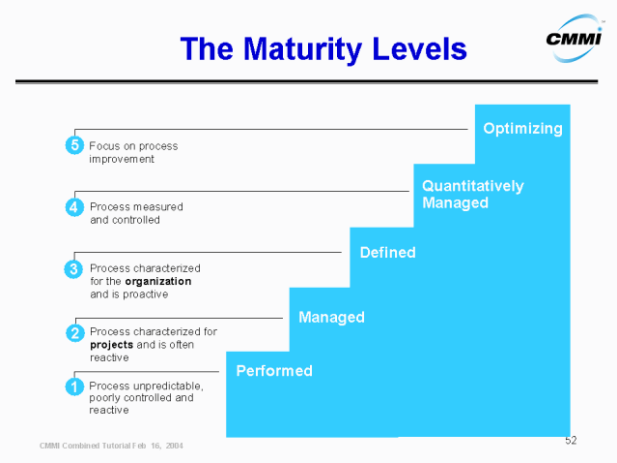
- Les données secrètes ne doivent pas être stockées dans le code.
- Les données secrètes ne sont pas stockées sauf si cela est indispensable. D'autres méthodes doivent être évaluées lors de la phase de conception.
- Les connexions à la base de données, les mots de passe, les clés ou les autres données secrètes ne doivent pas être stockés en texte clair.
- Les données sensibles ne doivent pas être journalisées en texte clair.
- Les données sensibles ne doivent pas être stockées dans des cookies ou transmises sous forme de chaîne de requête ou de zone de formulaire.

Gestion des exceptions

- Seul un minimum d'information doit être divulgué en cas d'exception.
- Les données sensibles ne doivent pas être journalisées.

ANNEXE D : UTILISATION DE CMMI POUR AMÉLIORER LA SÉCURITÉ DES APPLICATIONS

Le Software Engineering Institute (SEI) de Carnegie Mellon a donné une définition du CMMI (Capability Maturity Model® Integration) sur le site http://www.sei.cmu.edu/cmmi/. Le CMMI a fait couler beaucoup d'encre, mais nous n'allons exposer ici que quelques concepts fondamentaux. Le site Web renvoie vers de nombreuses conférences et ouvrages détaillés sur le sujet, notamment sur les concepts TSP (Team Software Process) et PSP (Personal Software Process). Les meilleures pratiques CMMI permettent entre autres aux entreprises d'établir un lien explicite entre les activités de gestion et d'ingénierie et les objectifs métier, et de propager la portée et la visibilité jusque dans le cycle de vie du produit et les activités techniques, afin de satisfaire les attentes du client en matière de produit et de service.



Il est important de garder à l'esprit les concepts CMMI lors de la mise en œuvre de modifications dans votre entreprise. En matière de modifications, une évolution est plus facile à mettre en œuvre qu'une révolution. Le niveau Optimisation est le plus prévisible mais il est aussi le plus coûteux. Un principe intéressant revendiqué par les défenseurs CMMI est que plus vos processus logiciels sont matures, plus la détection et la résolution très précoces des erreurs applicatives sont probables. Le tableau 9 : Activités liées à la sécurité des applications définies par CMMI représente les activités standard pour chaque niveau.

Légende graphique

CMMI = CMMI
The Maturity Levels = Les niveaux de maturité
Optimizing = Optimisation
Quantitatively Managed = Gestion quantitative
Defined = Définition
Managed = Gestion
Performed = Exécution
Focus on process improvement = Priorité à l'amélioration de processus
Process measured and controlled = Processus mesurés et contrôlés
Process characterized for **the organization** and is proactive = Processus caractérisés pour l'entreprise, proactifs
Process characterized for **projects** and is often reactive = Processus caractérisés pour les **projets**, souvent réactifs
Process unpredictable, poorly controlled and reactive = Processus imprévisibles, mal contrôlés et peu réactifs

Optimisation	• Recherche de meilleures pratiques d'amélioration continue extraites des métriques des processus.
Gestion quantitative	• Des analyses coordonnées de la sécurité des applications sont effectuées à toutes les phases. • L'entreprise a publié des normes s'appliquant à un développement sûr des applications.
Définition	• Des analyses de sécurité sont effectuées lors des phases de développement de chacune des applications. • Tous les membres des équipes de l'application sont sensibilisés à la sécurité. • Une stratégie de sécurité des applications prédéfinie est en vigueur dans toute l'entreprise.
Gestion	• Les audits de sécurité sont effectués régulièrement. • Les équipes d'audit et d'assurance qualité utilisent des outils d'analyse automatiques avant le déploiement. • L'entreprise possède une charte de sécurité des applications, mais des exceptions sont autorisées.
Exécution	• Des équipes d'audit ou « de piratage éthique » exécutent régulièrement des tests d'infiltration de l'infrastructure. • L'entreprise possède une charte de sécurité des applications, mais sa mise en œuvre est limitée.

Tableau 9 : Activités de sécurité des applications définies par CMMI

ANNEXE E : CLASSIFICATION DES RISQUES AVEC LE MODÈLE DREAD DE MICROSOFT

L'un des problèmes posés par les systèmes de classification trop simples est que les membres de l'équipe n'arrivent en général pas à se mettre d'accord sur les classifications.
Pour vous aider à surmonter cette difficulté, ajoutez de nouvelles dimensions qui vous aident à déterminer la signification réelle d'une menace et son impact pour la sécurité. Microsoft a élaboré son modèle DREAD pour vous aider à calculer les risques. Avec DREAD, vous obtenez la classification du risque d'une menace particulière en vous posant les questions suivantes :

- **Potentiel de nocivité :** quelle serait l'ampleur du dommage si la vulnérabilité était exploitée ?
- **Reproductibilité :** l'offensive est-elle facile à reproduire ?
- **Vulnérabilité :** est-il facile de lancer une offensive ?
- **Utilisateurs affectés :** en gros, combien d'utilisateurs seraient-ils concernés ?
- **Facilité de détection :** est-il facile de détecter la vulnérabilité ?

Une définition claire de ce que chaque critère représente pour votre système de classification évite toute confusion. Le tableau 10 représente un exemple d'une table de classification que les membres de l'équipe peuvent utiliser pour établir les différentes priorités des menaces.
Vous pouvez envisager d'utiliser une échelle plus précise (de 1 à 10) pour appliquer cette classification à toute l'entreprise.

Classification	Elevée (3)	Moyen (2)	Faible (1)
Potentiel de nocivité	L'agresseur peut corrompre la sécurité du système, se procurer une autorisation entièrement sécurisée, s'arroger le rôle de l'administrateur et envoyer du contenu.	Divulgation d'informations sensibles	Divulgation d'informations sans valeur
Reproductibilité	L'offensive peut être reproduite à tout moment et ne nécessite pas l'observation d'un créneau particulier.	L'offensive peut être reproduite, mais seulement dans un créneau ou une situation de durée très réduite.	L'offensive est très difficile à reproduire, même si la faille de sécurité est connue.
Vulnérabilité	Un programmeur néophyte pourrait lancer l'offensive en peu de temps.	Un programmeur compétent pourrait lancer l'offensive, puis la renouveler.	L'offensive ne peut être, à chaque fois, que le fait d'un agresseur extrêmement compétent et ayant des connaissances approfondies.
Utilisateurs affectés	Tous les utilisateurs, configuration par défaut, principaux clients	Certains utilisateurs, configuration autre que par défaut.	Très petit pourcentage d'utilisateurs, fonctionnalité obscure, affectant des utilisateurs anonymes.

Facilité de détection	La description de l'offensive est publiée. La vulnérabilité existe dans une fonction très souvent utilisée et se remarque facilement.	La vulnérabilité est une partie du produit rarement utilisée et seuls quelques utilisateurs risquent d'y être confrontés. Une analyse mûrement réfléchie est nécessaire pour détecter une utilisation malveillante possible.	L'erreur est obscure et il est peu probable que les utilisateurs puissent l'exploiter à des fins nuisibles.

Table 10 : Tableau de classification du modèle DREAD

Le tableau 11 est une classification DREAD possible de l'utilisation supposée de la « faille de réinitialisation du mot de passe d'un compte T-mobile **»** dans l'affaire du piratage du téléphone portable de la star Paris Hilton, qui avait débouché sur la publication sur le Web de la totalité de son répertoire téléphonique, notes et messages vocaux. Les mots clés sont mis en gras pour des raisons de clarté et montrent que même si une classification plus fine de 1 à 10 était utilisée, cette offensive resterait classifiée comme étant de dangerosité très élevée, voire imparable. La réaction de T-Mobile avait d'ailleurs été de désactiver la fonction concernée jusqu'à correction de l'erreur.

Critères	Classification	Commentaires
Dommage causé	3	L'offensive a permis de consulter et de gérer la **totalité** du compte de l'abonnée.
Reproductibilité	3	L'offensive fonctionne **à chaque fois**.
Mode d'exploitation	3	L'opération de copier–coller **s'effectue via le navigateur.**
Utilisateurs affectés	3	**Tous** les utilisateurs de T-Mobile en ligne.
Diffusion	3	La faille est **massivement commentée dans les médias** et des instructions décrivant son exécution sont publiées sur des sites Web bien indexés.
	15	**ÉLEVÉ**

Tableau 11 : Classification DREAD de la « faille de réinitialisation du mot de passe d'un compte T-mobile » : 5-7 : faible ; 8-11 : moyen ; 12-15 : élevé

Classifications Elevé, Moyen, Faible

Vous pouvez utiliser une classification simple de type Élevé, Moyen, Faible pour classer les menaces par priorités. Si une menace obtient la classification Élevé, elle pose un risque important pour votre application et doit être résolue dans les plus brefs délais. Les menaces de niveau Moyen doivent aussi être résolues, mais sont moins urgentes. Vous pouvez par contre vous permettre de ne pas tenir compte des menaces de niveau Faible, selon le travail et le coût nécessaire à leur résolution.

Autres ressources de Microsoft

Microsoft propose de nombreux modèles d'évaluation des menaces, dont un outil téléchargeable gratuitement appelé « Threat Model Tool » qui fait appel à la méthode STRIDE de classification par catégories.

ANNEXE F : TEST DE LA SÉCURITÉ DÉTERMINÉ PAR LES ÉVÉNEMENTS

Le test du système est un processus qui dépend des événements.

1) L'utilisateur envoie une requête.
2) Votre application répond.
3) La réponse est comparée avec la réponse anticipée ou préalablement mémorisée.
4) L'opération échoue ou réussit.

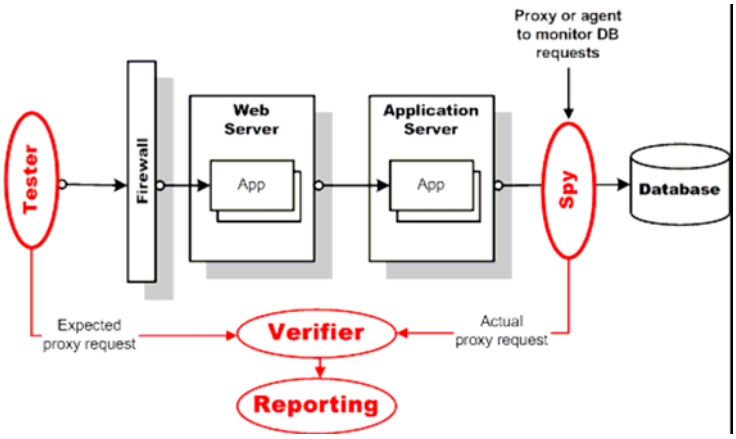
Les tests du système sont très intéressants lorsque vous disposez d'un environnement contrôlé et connaissez les requêtes utilisateurs possibles et leurs réponses.

Mais que se passe-t-il si votre application ne contrôle pas tous les services qu'elle utilise ? Et si vous souhaitez vérifier qu'un utilisateur authentifié est réellement incapable d'accéder aux données d'un autre utilisateur ? Vous vous trouvez alors souvent réduit à recourir à des tests manuels. Voyez par exemple l'application simple décrite ci-après.

À un niveau générique, si vous souhaitez protéger des données sensibles, vous écrivez soit un test d'unité, soit un test du système pour vérifier que les données sont extraites de la base de données. Mais comment faire pour savoir si l'Utilisateur A peut récupérer les données de l'Utilisateur B ? Comment vérifier qu'une entrée utilisateur de qualité médiocre n'a pas de répercussions sur les processus dorsaux ? Un test manuel est une solution qui demande toujours beaucoup de temps et d'argent. Une meilleure idée consiste à créer un système de test qui soit intégré à l'application.

Le **Testeur** génère un test. Il sait parfaitement quelle doit être la requête du proxy ou de l'agent **Espion** et il informe le **Vérificateur**, qui compare alors la requête attendue avec la requête réellement reçue par l'**Espion** protégeant la base de données.

L'exemple représente une base de données comme composant dorsal, mais il est possible de défendre ainsi n'importe quel service, messagerie, application XML ou application existante.



L'implémentation du code d'étude des requêtes est dépendante de l'application. Les conceptions possibles du composant **Espion** peuvent être les suivantes : un objet d'accès aux données (DAO) factice, un proxy, un mouchard ou une classe héritant du service protégé. Le concept consiste à créer du code spécialement réservé au test et à l'insérer dans le flux de données. Le code inséré détecte le test que vous souhaitez exécuter et transmet les données si nécessaire via le système de test.

La coordination des objets de test représentés en rouge permet un contrôle complet et très fin de toute une gamme de tests et peut être effectuée avec seulement un test de boîte noire ou de boîte blanche.

Légendes du graphique

Texte rouge

Tester = Testeur
Verifier = Vérificateur
Reporting = Reporting
Spy = Espion

Texte noir

Firewall = Pare-feu
Web server = Serveur Web
Application server = Serveur d'application
App = App.
Expected proxy request = Requête proxy attendue
Actual proxy request = Requête proxy réelle
Proxy or agent to monitor DB requests = Proxy ou agent surveillant les requêtes de la base de données
Database = Base de données

Copyright d'IBM