DXC.technology

ENSIAS

Mémoire de projet de fin d'études

Pour l'obtention du titre

d'Ingénieur d'Etat en Informatique

Filière : Sécurité des systèmes d'information

Sujet :

Mise en place d'une méthodologie de développement sécurisée basée SDLC

Présenté et soutenu le 01 Juillet 2019 par :

GUEZRI Amine

Sous l'encadrement de :

M.IBRAHIMI Ayoub (DXC)

M.RAHMOUN Amine (DXC)

Mme.AJHOUN Rachida (ENSIAS)

Membres du Jury :

Mme. EL BAKKALI Hanan (Président)

M. DOUKKALI Abdelaziz (Examinateur)

Année universitaire : 2018-2019

Dédicaces

Je dédie ce travail :

A mes très chers parents ; pour leur soutien moral et matériel qui ont été toujours à mes côtés pour surmonter toutes les difficultés et qui ont fait preuve de compréhension, d'affectation et de tolérance, aucune expression ne saurait exprimer les sentiments que je ressens envers eux.

....

A mes amis en souvenir de chaque moment de joie, gaieté et bonheur et à toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

Remerciements

Il m'est agréable de m'acquitter d'une dette de reconnaissance auprès de toutes les personnes dont l'intervention au cours de ce projet a favorisé son aboutissement.

Ainsi, je tiens à remercier tout le personnel de DXC technology pour son soutien et pour sa générosité considérable quant à l'offre de l'information.

Mes très chers remerciements vont à **Mr. Ibrahimi Ayoub** et **Mr. Rahmoun Amine** mes encadrants au sein de DXC technology, qui n'ont pas manqué de me préparer les conditions favorables au bon déroulement du projet et également à mon binôme de projet **Benfettah Oussama**.

Mes remerciements les plus sincères vont aussi au **Pr. Ajhoun**, mon encadrant à l'ENSIAS, pour les conseils qu'il m'a prodigués et pour son judicieux encadrement.

Je tiens également à adresser mes plus sincères remerciements à l'ensemble du corps enseignant de l'ENSIAS, pour avoir porté un vif intérêt à notre formation, et pour avoir accordé le plus clair de leur temps, leur attention et leur énergie et ce dans un cadre agréable de complicité et de respect.

Enfin, que tous ceux et celles qui ont contribué de près ou de loin à l'accomplissement de ce travail trouvent l'expression de mes remerciements les plus chaleureux.

Résumé

De nos jours, les applications informatiques sont devenues omniprésentes et nous les utilisons au quotidien. Le revers de la médaille est que ces applications étant disponibles de partout et utilisées par tout le monde, elles sont particulièrement exposées aux attaques des personnes malveillantes.

Plusieurs solutions sont suggérées pour remédier aux attaques auxquelles sont confrontés les logiciels mais leur déploiement est très coûteux. C'est dans ce cadre que s'inscrit ce projet de fin d'études : Produire des logiciels sécurisés tout en minimisant les coûts. Pour ce faire, nous allons mettre en place une méthodologie de développement sécurisé basée sur le cycle de vie du développement de logiciel SDLC de notre organisme d'acceuil.

Cette méthodologie est un ensemble d'activités obligatoires dans le domaine de la sécurité, réparties sur l'ensemble des phases du développement logiciel. Ces activités reposent sur deux concepts fondamentaux : La sensibilisation des équipes de développement et l'intégration des mesures de sécurité (Ces mesures sont définites en se basant sur la connaissance préalable des attaques logiciel).

Mots clés : SDLC, SDL, SAST, DAST, Fuzzing, Revue de code, Pentesting, Gestion de risque, Threat Modeling ...

Abstract

Nowadays, computer applications have become ubiquitous and we use them on a daily basis. The flip side is that these applications are available from everywhere and used by everyone, they are particularly vulnerable to attacks by malicious people.

Several solutions are suggested to remedy the attacks that software faces, but their deployment is very expensive. It is within this framework that this end-of-studies project is: Producing secure software with minimal cost. To do this, we will implement a secure development methodology based on the SDLC software development life cycle of our host organization.

This methodology is a set of mandatory activities in the field of security, spread over all phases of software development. These activities are based on two fundamental concepts: Training (or awareness) and the security measures defined based on prior knowledge of software attacks.

Keywords : SDLC, SDL, SAST, DAST, Fuzzing, Code Review, Pentesting, Risk Management, Threat Modeling ...

Table des figures

Liste des abréviations

CAWE : Comman Architectural weakness Enumeration.

CSRF : Cross Site Request Forgery.

CWE : Common weakness enumeration.

DAST : Dynamic Application security testing.

DDS : Docuement de conception.

DoS : Déni de service.

HSTS : Strict Transport Security.

IA : Assurance de l'information.

ISO : Organisation internationale de normalisation.

LDAP : Lightweight Directory Access Protocol.

MVC : Model View Controller.

OWASP : Open web application security Project.

SANS : 'SysAdmin, Audit, Network, Security' Organisation.

SAST : Static Application security testing.

SDLC : Software Development Life cycle.

SDL : Secure Development Lifecycle.

SRS : Spécification des exigences logicielles.

SQL : Structed Query Language.

WAF : Web application firewall.

WIVET : Web Input Vector Extractor Teaser.

XSS : cross-site scripting.

Table des matières

Introduction générale

La plupart des logiciels existants sont conçus avec peu de considération quant à la sécurité des informations, ce qui les rend vulnérables à de nombreuses menaces. Les patch sont l'une des solutions suggérées pour remédier aux attaques auxquelles sont confrontés les logiciels, mais en plus qu'ils ne résolvent pas les faiblesses de base introduites dans la phase de la conception, **leur déploiement est coûteux**.

Par conséquent, l'intégration des principes de sécurité dès les premières étapes du cycle de développement logiciel (SDLC) constituerait une meilleure solution et permettrait de produire un système plus cohérent.

Développer un système sécurisé nécessite une bonne conception globale qui prend en compte **la sécurité dès le début**. C'est dans ce cadre que s'inscrit ce projet de fin d'études qui a pour objectif de réaliser une méthodologie qui incorpore les principes de sécurité à chaque étape du cycle SDLC. Ces principes sont définis en fonction de la connaissance préalable des attaques de logiciels. Il faut s'assurer que chaque étape est conforme aux principes de sécurité à travers des tests.

Ce travail est structuré en quatre chapitres, dont voici une brève présentation :

Dans le premier, nous présentons le contexte général du projet. Nous allons annoncer la problématique, définir les objectifs de notre projet et décrire la démarche que nous avons suivi pour atteindre ces objectifs.

Dans le deuxième chapitre, nous étudions le modèle du cycle de vie de logiciel SDLC. Nous allons mettre le point aussi sur les failles logiciels les plus courantes : Les principes que nous allons définir dans la méthodologie sont basés sur la connaissance préalable des attaques logiciels.

Le troisième est dédié à la partie réalisation. Nous allons mettre le point, tout d'abord,

sur la partie théorique de la solution « SDLC securisé », ensuite nous allons implémenter cette solution au sein de notre organisme d'acceuil.

Le dernier chapitre comporte quelques simulations, déduites de divers analyses (Statique, dynamique ...) pour mesurer l'effectivité de notre solution.

Enfin, nous clôturons notre travail par une conclusion générale.

Chapitre 1

Contexte général du projet

Le but de ce chapitre est de situer le projet dans son environnement contextuel et organisationnel ; ce qui a amené impérativement à mettre en contexte l'environnement général du projet basé sur l'organisme d'accueil, la problématique du projet, ses objectifs ainsi que la démarche pour son déroulement.

1.1 Présentation de l'organisme d'accueil

Dans cette section, nous allons présenter une description détaillée de l'organisme d'accueil. Notamment, l'historique, le marché, la clientèle, état de concurrence, la structure RH, la culture et les partenaires de DXC technology.

1.1.1 Description général de l'organisme d'accueil

DXC Technology est une société de service américaine issue de la fusion de CSC (Computer Sciences Corporation) et de la division Enterprise Services de HPE (Hewlett-Packard Enterprise). CSC vient des métiers du conseil, tandis qu'HPE vient des métiers de l'infogérance et de l'infrastructure. DXC se revendique comme étant la première société de services informatiques indépendante au monde. Avec sa devise "Thrive on change", DXC se positionne dans le domaine de la transformation numérique auprès des entreprises et des gouvernements. L'entreprise compte 137 000 salariés et 6 000 clients, répartis dans plus de 70 pays.

Les informations indiquées ci-dessus sont des informations qui décrivent l'entreprise d'une manière générale et non pas la filiale DXC Technology Maroc.

Précédemment connue sous le nom HP CDG IT Services Maroc, DXC Technology MAROC, implantée depuis 2007, est une joint-venture entre deux grands groupes de renommée : DXC Technology qui détient 51% de la société et la Caisse de Dépôt et de Gestion qui détient 49% de la société. DXC Technology au Maroc a pour vocation d'accompagner les très grands comptes et donneurs d'ordre publics et privés dans leur transformation digitale et d'adresser ainsi l'ensemble des enjeux IT que sont l'Infogérance, la Modernisation Applicative, Business Intelligence & Analytics, Workplace & Mobilty Services, Business Process Services, la Cyber Sécurité et le Conseil. DXC Technology réalise un chiffre d'affaire de 257 Millions de Dirhams en 2017 **[W1]**.

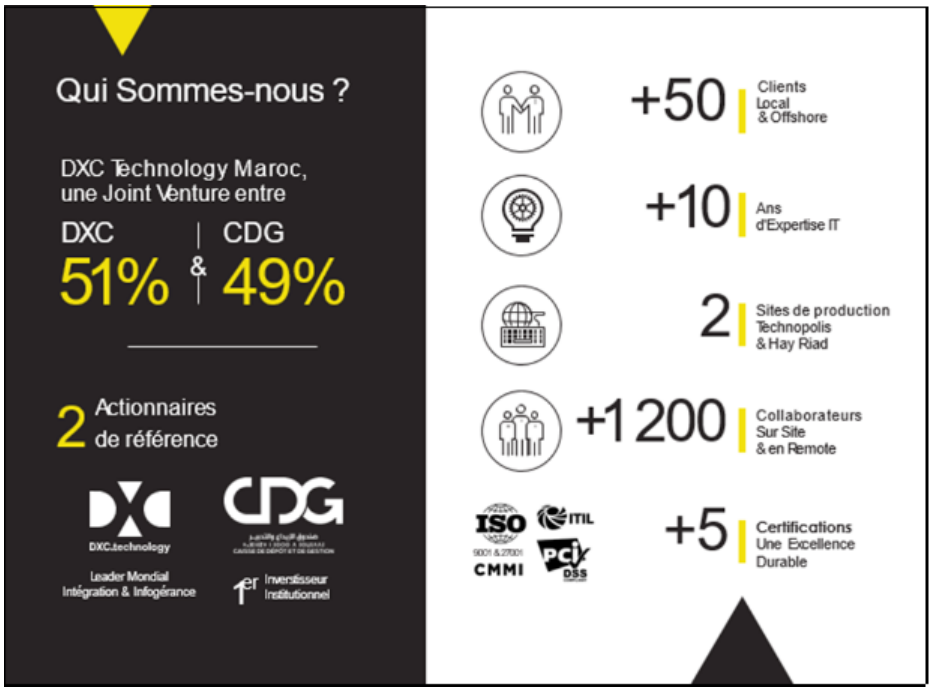


FIGURE 1.1 – Fiche technique DXC Technology Maroc

1.1.2 Domaines d'activités

DXC Technology Maroc offre un portefeuille complet de services et accompagne les entreprises pour être disruptives, se réinventer, développer de nouvelles agilités, conduire les projets de la stratégie à l'implémentation et faire du changement un catalyseur de succès "Thrive On Change". La partie suivante montre en détails les services fournis par l'entreprise **[W2]**.

- Data center : Hébergement, support & supervision technique, Cloud privé/public
- Cloud & Infrastructure : Cloud Privé, Public, Plateforme Traditionnelle
- Service Applicatif : Intégration, Modernisation, Migration, TMA/ TRA
- BI & Analytics : Plateformes Analytiques sur Site, Cloud
- Workplace & Mobility : Digital Worplace/ Automation, User-Centric, Business-Desk
- Business Process : Front, Back Office
- Cyber Security : Données, Applications, Infrastructure, Endpoints
- Conseil : Accélération de la transformation digitale

1.1.3 Organigramme de l'entreprise

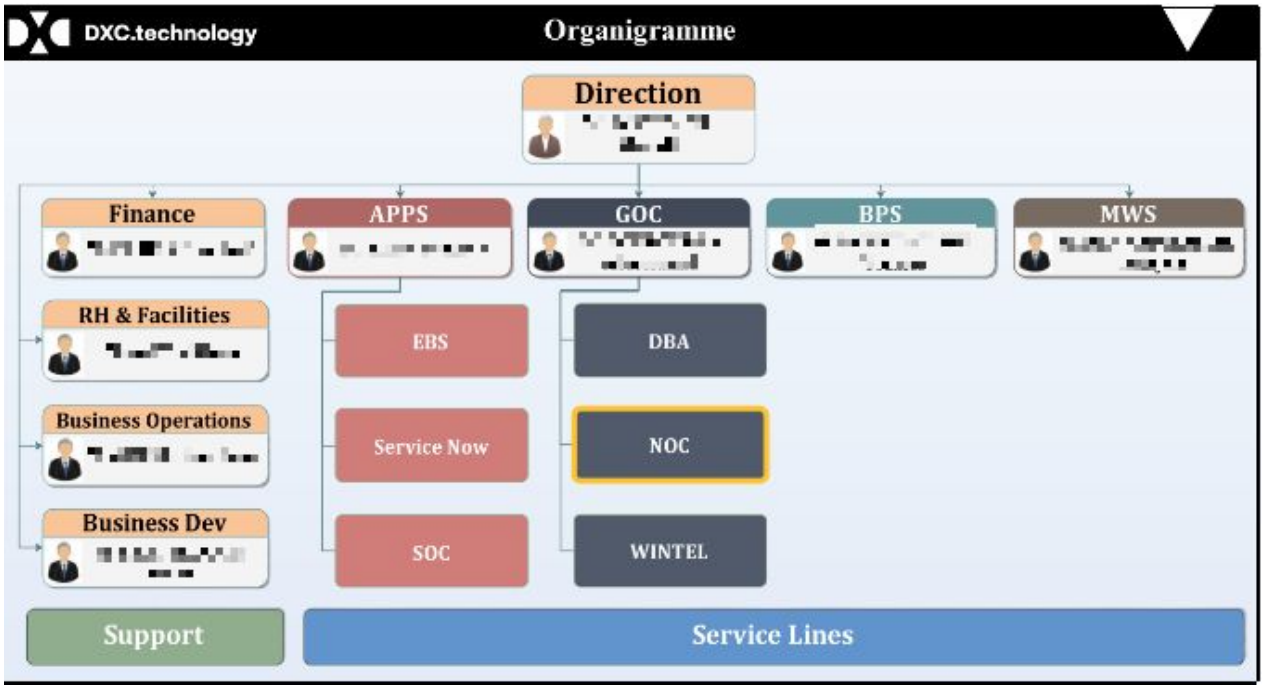


FIGURE 1.2 – Organigramme DXC Technology Maroc

L'organigramme ci-dessus décrit quelques divisions principales de l'entreprise et pas la totalité des divisions, il est divisé en 2 parties "Support" et "Services Lines" :

• Support : rassemble l'ensemble d'activités de gestion considérées comme ne constituant pas le cœur de métier. Ces activités assurent le fonctionnement de l'entreprise et sont généralement communes à plusieurs divisions ou ligne de produits métier, tel que le service Finance, Ressources Humaines, marketing et service de ventes.

• Service Lines : contient les divisions qui représentent le cœur de métier, ces divisons sont :

1- APPS : (Application Services) représente généralement les services de développement informatique et qualité logiciel, ainsi que l'intégration des ERP et le support applicatif.

2- GOC : (Global Operations Services) c'est une division dédié pour les services d'Infrastructure, Cloud et Sécurité des systèmes d'information.

3- BPS : ou BPO (Business Process Outsourcing) l'ensemble des activités qui ont comme but l'externalisation d'une partie de l'activité de l'entreprise vers un prestataire extérieur.

4- MWS : assure les services de gestion et support de mobilité, et le support au Delivery.

Dans la département APPS, il a l'équipe SOC (Security Operations Center) sous la supervision de Mr. Ayoub IBRAHIMI notre encadrant.

L'équipe SOC s'occupe de plusieurs missions de sécurité informatique chez des clients finaux, ces missions s'étalent sur plusieurs domaines comme :

- Le monitoring et l'administration des solutions de sécurité comme les solutions SIEM, Antivirus, Firewall, WAF, Proxy, etc.
- Le consulting.
- L'analyse comportementale basée sur l'intelligence artificielle.
- La veille de la sécurité informatique

1.2 Présentation du projet

Une bonne définition du cadre du projet s'avère particulièrement importante dans le déroulement du projet pour atteindre les résultats escomptés. Cette section comporte une description du projet, ses objectifs et finalement sa planification.

1.2.1 Problématique

Auparavant, les parties prenantes du produit logiciel ne considéraient pas que la sécurité du logiciel était une priorité élevée. On pensait qu'une infrastructure de réseau sécurisée fournirait le niveau de protection nécessaire contre les attaques malveillantes. Dans l'histoire récente, beaucoup d'utilisateurs malveillants ont réussi à pénétrer dans des canaux d'authentification valides grâce à de telles techniques : Scripts intersites, injection SQL (Structured Query Language) et débordement de la mémoire tampon. Le groupe Gartner indique que **plus de 70% des vulnérabilités des entreprises se trouvent dans les applications logicielles** plutôt que dans les limites du réseau (Aras, Barbara

et Jeffrey, 2008).

De ce fait, l'assurance des logiciels est devenue rapidement un domaine d'intervention de l'Assurance de l'information (IA) dans les secteurs financier, gouvernemental et de la fabrication afin de réduire les risques de code non sécurisé.

La plupart des organisations adoptent un cycle de développement de logiciel (SDLC) pour la mise en œuvre de systèmes informatiques. SDLC est un processus de cycle de vie en plusieurs étapes destiné à fournir des systèmes informatiques garantissant la qualité tout en répondant aux spécifications dans les délais et les coûts estimés.

Le « Systems Sciences Institute » d'IBM a indiqué que le coût de la réparation d'un bogue découvert au cours de la phase de mise en œuvre **était environ onze fois plus élevé** que celui identifié lors de la conception **[W3]**. En bref, le coût de réparation d'un bogue augmente de façon exponentielle à mesure que le logiciel progresse le long du SDLC (Voir la figure ci-dessous).

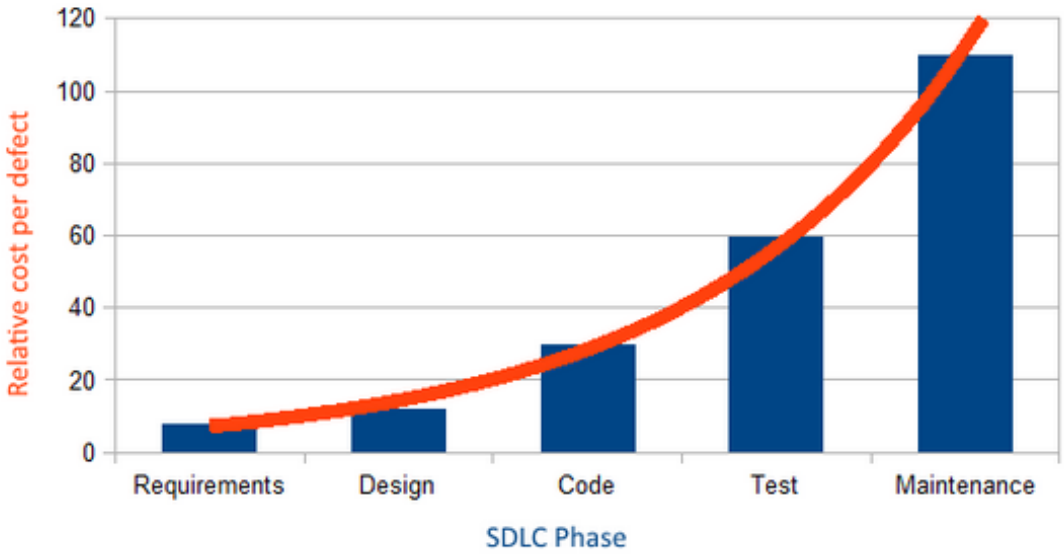


FIGURE 1.3 – Coût de réparation des failles durant le cycle de devloppement logiciel

Le dicton populaire "**Mieux vaut prévenir que guérir**" est vrai dans le développement de logiciels car il devient généralement plus difficile de résoudre un problème à mesure que le produit approche de la fin du SDLC. La réparation des bogues [1] introduits lors de la phase de conception coûte plus cher, s'ils n'étaient pas traité correctement ; car ils sont plus complexes à résoudre. Les modifications de code pour un correctif de bogue peuvent également affecter la fonctionnalité de l'application, nécessitant encore plus de contre-modifications, augmentant le coût, le temps et les efforts.

Dans une banque, par exemple, où les produits sont potentiellement utilisés par des milliers, la réparation d'une faille de sécurité identifiée une fois l'application lancée coûte très chère. Les efforts, le temps et l'argent nécessaires pour résoudre le problème sont nettement plus importants que si la faille avait été identifiée et résolue au cours des premières étapes de développement. De plus, la complexité du déploiement et de la mise en œuvre des modifications dans un environnement de production réel augmenterait encore les coûts globaux.

1.2.2 Objectifs du projet

Les bogues sont inévitables, mais l'intégration « en amont » de la sécurité dans le cycle de développement permet de créer un logiciel plus fiable. Un SDLC traditionnel effectue des tests de sécurité à la fin, une fois que les fonctionnalités requises de l'équipe de développement sont en place. Cependant, les tests de sécurité et la prise en compte des facteurs de risque au cours des phases de développement **éviteront les bogues et les failles de sécurité lors des étapes ultérieures**. De ce fait, l'objectif de notre projet est de trouver un moyen d'intégrer les mesures de sécurité dès la première phase de notre cycle de développement. Comme ça, nous aurons assuré une sécurité applicative optimale avec des coûts acceptables.

1. Les failles de sécurité appartiennent à la catégorie des Bogues. Un bogue est un défaut dans le logiciel qui cause un dysfonctionnement. Une faille de sécurité est un défaut mineur qui ne provoque pas de dysfonctionnement en utilisation courante, **mais** permet à un utilisateur malicieux ou un logiciel malveillant d'effectuer des opérations non autorisées à partir d'un exploit.

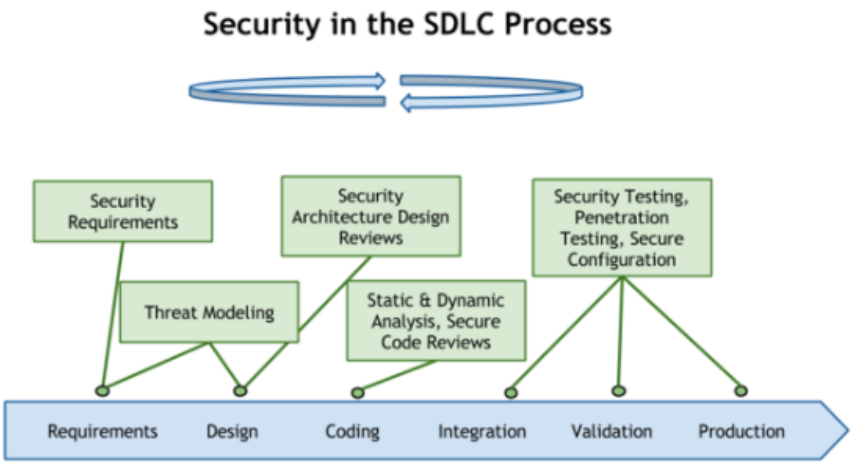


FIGURE 1.4 – Integration de la sécurité en amont dans le cycle de développement **[W4]**

1.2.3 Planification du projet

La planification est une phase très importante de chaque projet ; elle donne une idée claire en ce qui concerne le déroulement du projet et pour déterminer son périmètre.

L'objectif du projet est la mise en place d'une méthodologie de développement sécurisée basée sur le cycle de développement SDLC.

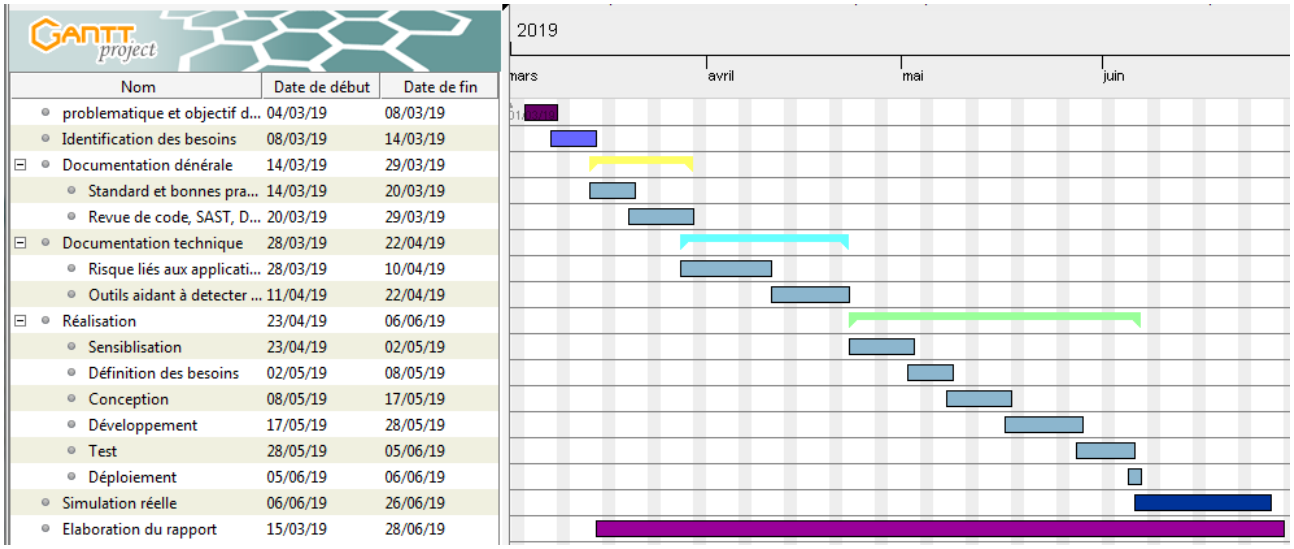
Notre mission consistait tout d'abord à se bien documenter sur les différents standards et bonnes pratiques de sécurité ainsi que sur les techniques de revues de code. D'autant plus, la phase de documentation inclut aussi une étude globale portant sur l'ensemble des erreurs de développement, de conception et de configurations existantes aujourd'hui (cette étude nous a bien été utile en l'enrichissement de notre savoir en terme de sécurité informatique), et d'une étude technique qui consiste à réaliser un ensemble de Benchmark (Threat Modeling, les outils de SAST, DAST et Fuzzing ...).

La phase de documentation était aussi utile en la réalisation d'une méthodologie que les équipes de développement vont suivre pendant le cycle de développement de logiciels. Cette méthodologie a pour but d'**améliorer la sécurité de nos applications à moindre coût**. Ce projet a donné naissance à plusieurs livrables, à intégrer au niveau des différentes phases du cycle de développement SDLC, que nous avons jugés primordiales à

la production de logiciel sécurisé.

Enfin, Nous avons réalisé des simulations réelles pour mesurer l'effectivité de notre méthodologie.

Gantt prévisionnel :



Nom	Date de début	Date de fin
problematique et objectif d...	04/03/19	08/03/19
Identification des besoins	08/03/19	14/03/19
Documentation dénérale	14/03/19	29/03/19
Standard et bonnes pra...	14/03/19	20/03/19
Revue de code, SAST, D...	20/03/19	29/03/19
Documentation technique	28/03/19	22/04/19
Risque liés aux applicati...	28/03/19	10/04/19
Outils aidant à detecter ...	11/04/19	22/04/19
Réalisation	23/04/19	06/06/19
Sensiblisation	23/04/19	02/05/19
Définition des besoins	02/05/19	08/05/19
Conception	08/05/19	17/05/19
Développement	17/05/19	28/05/19
Test	28/05/19	05/06/19
Déploiement	05/06/19	06/06/19
Simulation réelle	06/06/19	26/06/19
Elaboration du rapport	15/03/19	28/06/19

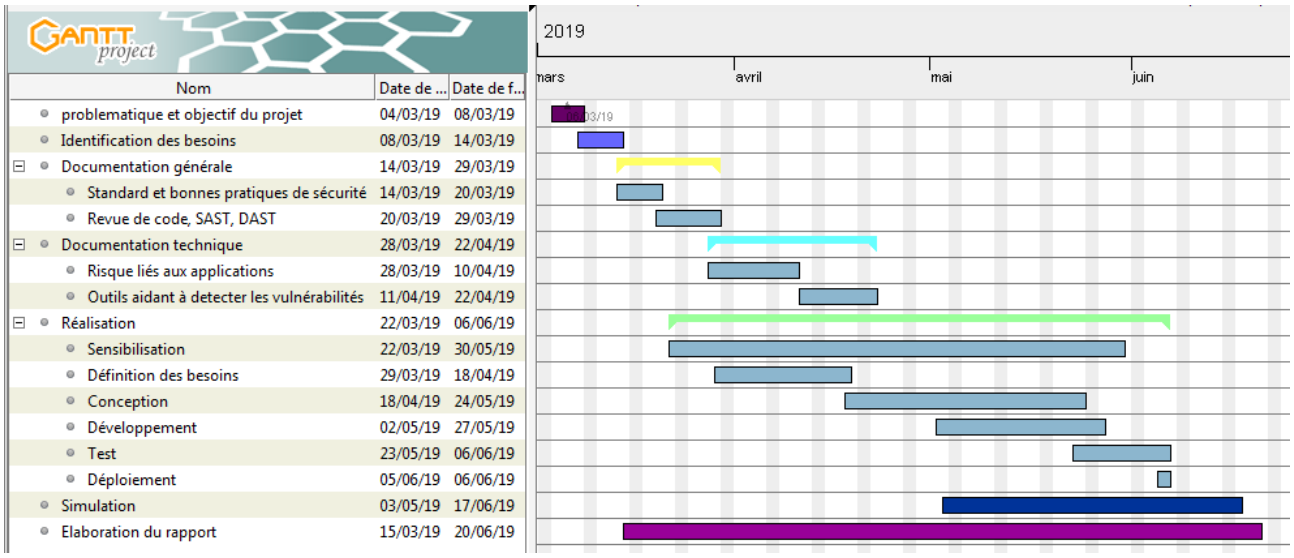
FIGURE 1.5 – Diagramme de Gantt prévisionnel

En gros le projet devra s'appliquer au planning prévisionnel présenté dans la figure 5.

En cours d'exécution, plusieurs changements ont été effectués pour bien suivre le bon déroulement du projet. Dans la phase de réalisation, des tâches ont pris plus de temps que prévue.

En tant que binome, nous avons réparti le travail entre nous d'où le prallélisme dans le diagramme de gantt suivant.

Gantt exécutée :



Nom	Date de ...	Date de f...
problematique et objectif du projet	04/03/19	08/03/19
Identification des besoins	08/03/19	14/03/19
Documentation générale	14/03/19	29/03/19
Standard et bonnes pratiques de sécurité	14/03/19	20/03/19
Revue de code, SAST, DAST	20/03/19	29/03/19
Documentation technique	28/03/19	22/04/19
Risque liés aux applications	28/03/19	10/04/19
Outils aidant à detecter les vulnérabilités	11/04/19	22/04/19
Réalisation	22/03/19	06/06/19
Sensibilisation	22/03/19	30/05/19
Définition des besoins	29/03/19	18/04/19
Conception	18/04/19	24/05/19
Développement	02/05/19	27/05/19
Test	23/05/19	06/06/19
Déploiement	05/06/19	06/06/19
Simulation	03/05/19	17/06/19
Elaboration du rapport	15/03/19	20/06/19

FIGURE 1.6 – Diagramme de Gantt exécutée

Chapitre 2

Le cycle de développement de logiciel SDLC

Dans ce chapitre nous allons introduire de façon générale la sécurité informatique, ensuite nous allons présenter les différentes phases du cycle de développement de logiciel SDLC, et finalement nous allons mettre le point sur les différents standards et bonne pratiques de sécurité.

2.1 Introduction à la sécurité informatique

Les systèmes d'information prennent de plus en plus une place stratégique au sein des entreprises. Ainsi la notion du risque lié à ces derniers devient une source d'inquiétude et une donnée importante à prendre en compte, ceci en partant de la phase de conception d'un système d'information jusqu'à son implémentation et le suivi de son fonctionnement.

Les pratiques associées à la sécurité des systèmes d'information constituent un point à l'importance croissante dans l'écosystème informatique qui devient ouvert et accessible par utilisateurs, partenaires et fournisseurs de services de l'entreprise. Il devient essentiel pour les entreprises de connaître leurs ressources en matière de système d'information et de définir les périmètres sensibles à protéger afin de garantir une exploitation maîtrisée et raisonnée de ces ressources.

En général, quand on parle de la sécurité, on spécifie sur quel niveau elle est implantée. Pour faire simple je vais énumérer les 3 niveaux les plus connus :

• **Sécurité réseau** : dans ce cas, on cherche à instaurer la sécurité au niveau d'un réseau (ou sous-réseau) entier de telle sorte à empêcher toute atteinte à n'importe quel élément faisant partie de ce réseau là. Les firewalls et les NIPS (Network-based Intrusion Prevention System) sont des solutions souvent envisagées dans cette situation.

• **Sécurité système** : dans ce cas de figure, le système et les services tournés sur un ordinateur (serveur, station de travail, terminal...) est ciblé par la sécurité. Les anti-virus constituent une solution très populaire, mais aussi les HIPS (Host-based Intrusion Prevention System) ou HIDS (Host-based Intrusion Detection System).

• **Sécurité applicative** : Ici ce n'est ni le réseau ni le système entier d'un ordinateur qui nous intéresse, mais un logiciel ou une application en particulier (site, application web...). Dans ce cas de figure on peut utiliser des correctifs, des filtres, des IPS applicatifs, des WAF (Web Application Firewall)...

2.1.1 Sécurité applicative

Selon ISO 27034 : La sécurité des applications est un processus effectué pour appliquer des contrôles et des mesures aux applications d'une organisation afin de gérer le risque de leur utilisation. Les contrôles et les mesures peuvent être appliquées à l'application elle même (ses processus, les composants, les logiciels et résultats), à ses données (données de configuration, les données de l'utilisateur, les données de l'organisation), et à toutes les technologies, les processus et acteurs impliqués dans le cycle de vie de l'application **[W5]**.

Nous avons choisi de travailler principalement sur la sécurité applicative parceque nous avons constatés, selon nombreuses statistiques que nous allons cités par la suite, que la plus part des attaques ciblent la couche applicative.

75% des vulnérabilités se retrouvent dans la couche applicative **[W6]**.
84% des attaques ciblent la couche applicative **[W6]**.
70% des applications contient au moins une vulnérabilité classifiée dans le top 10 OWASP **[W7]**.
15% des applications Web ont une vulnérabilité critique ou élevé **[W8]**.

2.1.2 Retombés d'une faille de sécurité

Les attaquants peuvent potentiellement utiliser différents chemins à travers votre application pour porter atteinte à votre métier ou à votre entreprise, exemple cité dans la figure ci-dessous. Chacun de ces chemins représente un risque qui peut, être suffisamment grave pour mériter votre attention. Parfois, ces chemins sont faciles à trouver et à exploiter, et parfois ils sont extrêmement difficiles. De même, le préjudice causé peut n'avoir aucune conséquence, ou faire cesser votre activité. Pour déterminer le risque pour votre entreprise, vous pouvez **évaluer la probabilité relative à chaque agent de menace** (Voir section 2, Chapitre 3), vecteur d'attaque, et vulnérabilité et les combiner avec une estimation d'impact technique et financier pour votre entreprise. Ensembles, ces facteurs

déterminent le risque global **[W9]**.

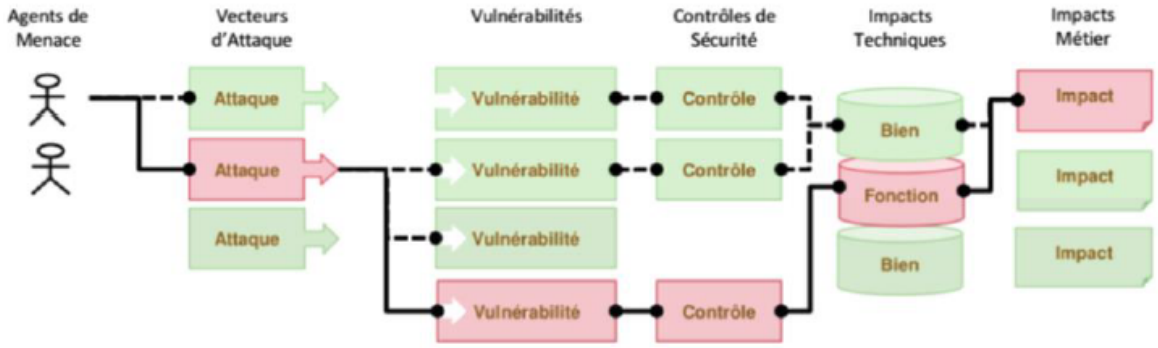


FIGURE 2.1 – Processus d'attaque d'une application

L'objectif de notre projet est d'integrer **la sécurité** en amont dans le cycle de développement de logiciel. Pour ce faire, il faut tout d'abord comprendre la logique du cycle et le rôle de chaque phase.

2.2 Le cycle de développment de logiciel SDLC

SDLC est un processus suivi pour un projet de logiciel, au sein d'une organisation de logiciel. Il consiste en un plan détaillé décrivant comment développer, entretenir, remplacer et modifier ou améliorer des logiciels spécifiques. Le cycle de vie définit une méthodologie pour améliorer la qualité des logiciels et le processus de développement global.

La figure suivante est une représentation graphique des différentes étapes d'un SDLC typique.

FIGURE 2.2 – Le cycle de développement de logiciel

2.2.1 Phase « Définition de besoin »

L'analyse des exigences est l'étape la plus importante et la plus fondamentale du cycle de vie de développement de logiciel. Il est effectué par les membres les plus expérimentés de l'équipe avec les contributions du client, du département des ventes, des études de marché et des experts de domaine du secteur. Ces informations sont ensuite utilisées pour planifier l'approche de base du projet et mener une étude de faisabilité du produit dans les domaines économique, opérationnel et technique.

La planification des exigences en matière d'assurance qualité et l'identification des risques associés au projet sont également effectuées dans cette phase. Le résultat de l'étude de faisabilité technique est de définir les différentes approches techniques qui peuvent être suivies pour mener à bien le projet avec un minimum de risques.

Une fois l'analyse des besoins effectuée, l'étape suivante consiste à définir et documenter clairement les exigences du produit et à les faire approuver par le client ou les analystes du marché. Cela se fait par le biais d'un document SRS (spécification des exigences logicielles) qui comprend toutes les exigences du produit à concevoir et à développer pendant le cycle de vie du projet.

À la fin de cette phase, nous aurons comme « **Output** » :

- Document de cadrage.
- Cahier de spécification.
- Contrat commercial (Engagement).

2.2.2 Phase « Architecture »

SRS est la référence pour les architectes de produits afin de proposer la meilleure architecture pour le produit à développer. Sur la base des exigences spécifiées dans SRS, plusieurs approches de conception pour l'architecture de produit sont proposées et documentées dans une spécification de document de conception DDS.

Cette DDS est examinée par toutes les parties prenantes importantes et sur la base de divers paramètres tels que l'évaluation des risques, la robustesse du produit, la modularité de la conception, le budget et les contraintes de temps, la meilleure approche de conception

étant choisie pour le produit.

Une approche de conception définit clairement tous les modules architecturaux du produit ainsi que sa représentation en matière de communication et de flux de données avec les modules externes et tiers (le cas échéant). La conception interne de tous les modules de l'architecture proposée doit être clairement définie avec le moindre détail dans DDS.

À la fin de cette phase, nous aurons comme « **Output** » :

- Rapport de faisabilité.
- Dossier d'architecture.

2.2.3 Phase « Développement »

Dans cette phase de SDLC, le développement réel commence et le produit est construit. Le code de programmation est généré selon DDS au cours de cette étape. Si la conception est effectuée de manière détaillée et organisée, la génération de code peut être accomplie sans problème.

Les développeurs doivent suivre les directives de codage définies par leur organisation et des outils de programmation tels que des compilateurs, des interprètes, des débogueurs, etc. sont utilisés pour générer le code. Différents langages de programmation de haut niveau tels que C, C ++, Pascal, Java et PHP sont utilisés pour le codage. Le langage de programmation est choisi en fonction du type de logiciel en cours de développement.

À la fin de cette phase, nous aurons comme « **Output** » :

- CD d'installation.
- Une documentation.

2.2.4 Phase « Test »

Cette étape est généralement un sous-ensemble de toutes les étapes car dans les modèles SDLC modernes, les activités de test sont principalement impliquées dans toutes les étapes de SDLC. Cependant, cette étape fait référence à la seule étape de test du produit où les défauts du produit sont signalés, suivis, corrigés et testés à nouveau, jusqu'à ce que

le produit atteigne les normes de qualité définies dans le SRS.

2.2.5 Phase « Déploiement»

Une fois le produit testé et prêt à être déployé, il est officiellement mis sur le marché. Parfois, le déploiement du produit s'effectue par étapes, conformément à la stratégie commerciale de cette organisation. Le produit peut d'abord être commercialisé dans un segment limité et testé dans un environnement professionnel réel (tests d'acceptation UAT-User).

Ensuite, en fonction des commentaires reçus, le produit peut être publié tel quel ou avec les améliorations suggérées dans le segment de marché ciblé. Une fois le produit commercialisé, sa maintenance est effectuée pour la clientèle existante.

À la fin de cette phase, nous aurons comme « **Output** » :

- Des patch/hotfix dans le cas de la maintenance.
- De nouveaux programmes d'installations dans le cas des évolutions.

Après avoir compris la logique du cycle de développement de logiciel, nous allons essayé de trouver un moyen pour integrer la sécurité dans notre cycle.

2.3 La sécurité dans le cycle de développement SDLC

Les vulnérabilités de sécurité peuvent être introduites dans n'importe quelle phase du cycle de développement de logiciel.

• **Définition de besoins** : Le manque des exigences de sécurité au niveau de la formulation des besoins.

• **Architecture** : Des failles de sécurité dû à des erreurs de conception.

• **Développement** : Des failles de sécurité dû à des erreurs de développement.

• **Test** : Absence de méthodologie de test.

• **Déploiement** : Absence d'une méthodologie de déploiement.

Dans cette section, nous allons se concentrer sur les deux phases 'conception et développement' puisqu'elles sont les deux phases dans les quelles s'introduisent le plus grand nombre de vulnérabilités.

Pour augmenter la sécurité durant ces deux phases, la seule solution rentable que nous avons pu trouver c'est de **prévoir** les erreurs qu'un architecte ou un développeur peut commettre. Cette prévention, nous aidera à formuler des recommandations de sécurité que nous allons délivrer par la suite aux equipes de conception et de développement sous forme de guide de sécurité.

Cette prévention nécessite :
- Des connaissances des failles de sécurité les plus courantes (**Section 2.3.1**).
- Des connaissances sur les erreurs de développement et de conception (**Section 2.3.2**).

Dans la section qui suit, nous allons énoncés les failles de sécurité les plus courantes.

2.3.1 Vulnérabilités et attaques web "OWASP TOP 10 vulnerabilities"

Open Web Application Security Project (OWASP) est une communauté en ligne travaillant sur la sécurité des applications Web. Sa philosophie est d'être à la fois libre et

ouverte à tous. Elle a pour vocation de publier des recommandations de sécurisation Web et de proposer aux internautes, administrateurs et entreprises des méthodes et outils de référence permettant de contrôler le niveau de sécurisation de ses applications Web.

2.3.1.1 Les injections

Le risque d'injection est placé au premier rang dans le classement OWASP TOP 10. L'impact technique et sa facilité d'exploitation sont, selon l'étude OWASP, à leur maximum. D'ailleurs, l'impact métier est généralement très critique car le risque est basé sur des supports primaires tels que les données utilisateur ou entreprise qui sont souvent confidentielles. Les dernières années n'ont pas été de tout repos pour les entreprises. Les fuites de données via injection de données sont devenues monnaie courante. Verizon avec 1,5 million de données utilisateurs, Yahoo ! avec 500 millions de comptes utilisateurs et mots de passe, Sony PSN avec 7 millions de cartes de crédit, Vtech, Visa, Adobe, la liste est longue.Il existe une multitude d'attaques par injection telles que LDAP, XPath, NoSQL, OS, etc. La plus connue est sûrement l'injection SQL, qui permet de tronquer des commandes SQL utilisées par l'application web afin de récupérer le maximum d'informations provenant d e bases de données.

Voici un graphique présentant l'évaluation du risque selon l'OWASP :

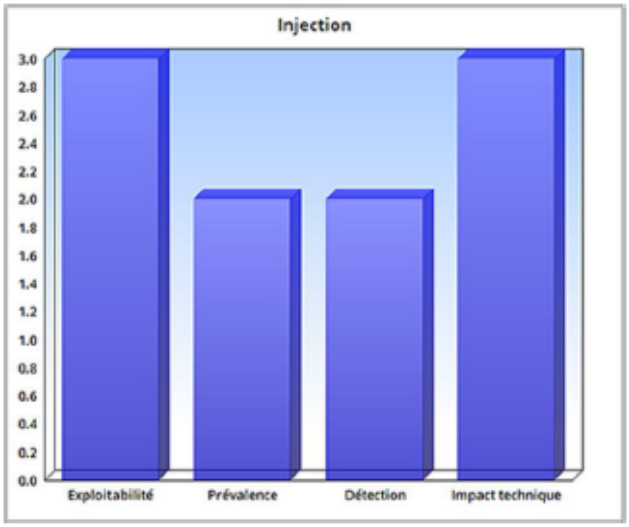


FIGURE 2.3 – Evaluation du risque de l'injection

2.3.1.2 La violation de gestion d'authentification et de session

La violation de session et d'authentification permet à un cybercriminel de capturer ou contourner les méthodes d'authentification utilisées par une application web. Les attaques de ce type sont dues à des défauts de conception lors de la création de l'application ou au manque de chiffrement réseau ou des mots de passe.

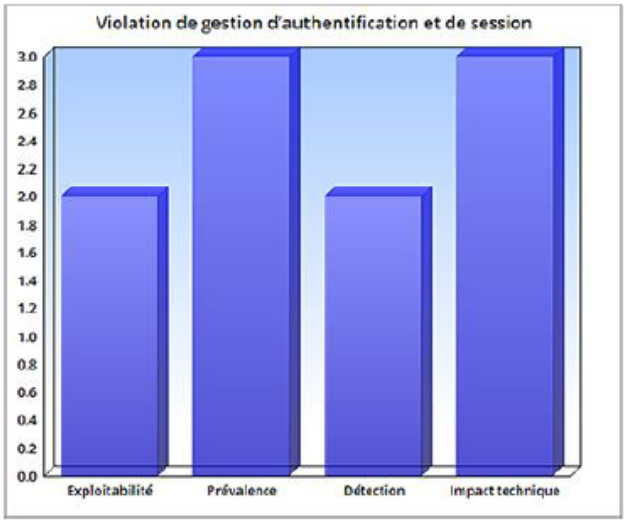


FIGURE 2.4 – Evaluation du risque de la violation de gestion d'authentification

2.3.1.3 Cross-Site Scripting XSS

La troisième place du classement OWASP TOP 10 est attribuée au Cross Site Scripting qui a pour particularité d'être le seul risque du TOP 10 ayant comme indice de prévalence "très répandu". Effectivement, d'après une étude de l'éditeur Egdescan, 52% des applications web seraient sujettes au XSS en 2015. Malgré les protections sur les navigateurs, les pare feu applicatifs (WAF) et les outils d'analy ses de code, les XSS restent une vraie source de menaces pour les utilisateurs. Le XSS permet à un cybercriminel d'envoyer du code JavaScript à l'aide d'une entrée utilisateur (formulaire, en tête HTTP, query string, etc.) mal protégée, afin de récupérer des identifiants de sessions, de hameçonner, d'enregistrer les touches frappées par l'utilisateur, etc.

Voici, d'après l'OWASP, l'évaluation du risque lié au XSS :

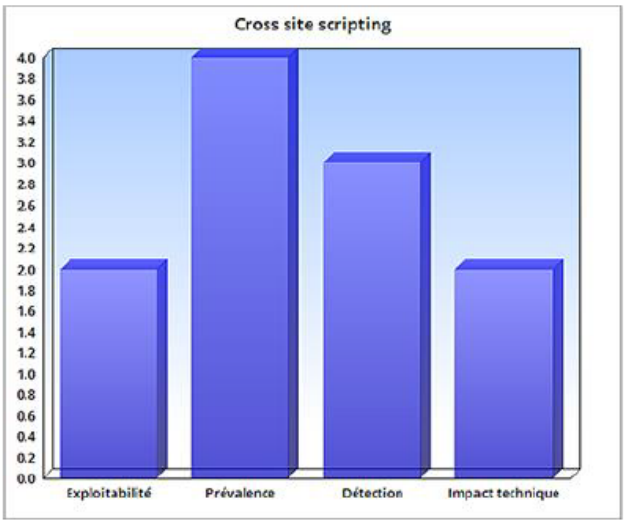


FIGURE 2.5 – Evaluation du risque du XSS

2.3.1.4 Références directes non sécurisées à un objet

Les références directes non sécurisées à un objet sont des vulnérabilités s'attaquant à la mauvaise gestion des droits et d'identification dans une application. Chaque site web contenant des données dynamiques comme un back office pour l'administration en arrière plan d'une application, doit être pourvu de rôles d'utilisateurs, d'administrateurs ou de modérateurs suivant les besoins de l'application. Si une référence à un objet n'est pas contrôlée à chaque instanciation de celui ci, un cybercriminel pourrait donc obtenir et modifier des information s auxquelles il ne devrait pas avoir accès. Une vulnérabilité du type Références directes non sécurisées a été découverte sur YouTube par un chercheur en sécurité russe nommé Kamil Hismatullin en 2015. Celui ci pouvait détruire n'importe quelle vidéo en mo difiant l'identifiant de la vidéo lors de la demande de suppression sur sa page YouTube. Même si l'identifiant de la vidéo ne lui appartenait pas, la vidéo était supprimée car la vérification de l'utilisateur n'était pas faite.

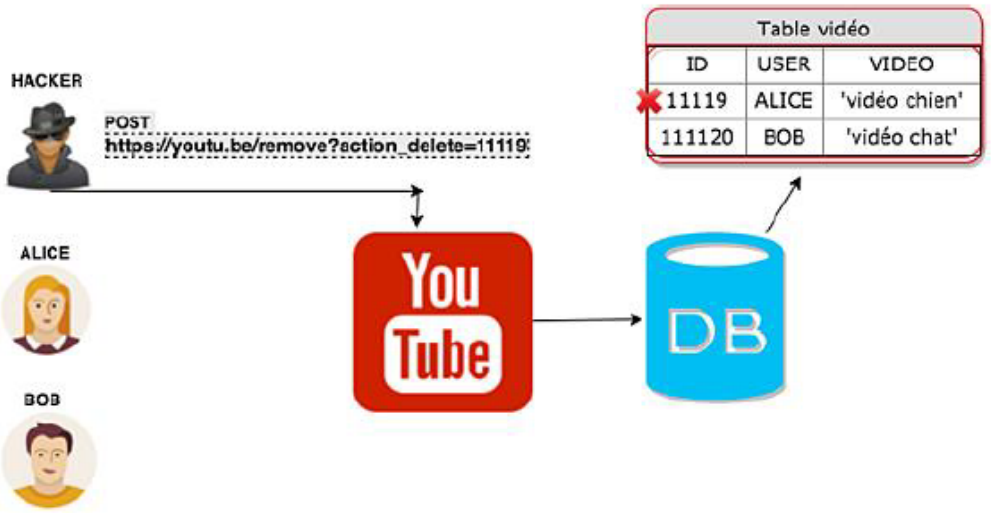


FIGURE 2.6 – Exemple de "Références directes non sécurisées à un objet"

2.3.1.5 Mauvaise configuration de sécurité

La mauvaise configuration des éléments de sécurité arrive en cinquième position du classement OWASP TOP 10. Le périmètre d'une mauvaise configuration de sécurité peut dépasser le périmètre d'une application. La mise à jour des serveurs web, de s librairies, des comptes administrateur inactifs, l'affichage des messages d'erreurs... sont des vulnérabilités associées à ce risque. Voici l'évaluation des menaces suivant l'OWASP :

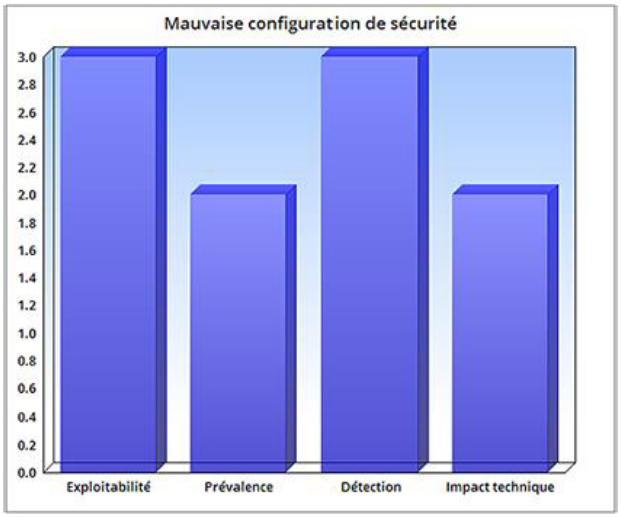


FIGURE 2.7 – Evaluation du risque de la mauvaise configuration de sécurité

2.3.1.6 Exposition de données sensibles

L'exposition de données sensibles concerne surtout les attaques cryptographiques, voire le manque de chiffrement sur les réseaux et données stockées. Comme vu dans le chapitre précédent (cf. Panorama de la sécurité web La sécurité des navigateurs et serveurs web), les attaques du type homme du milieu (MITM) sont assez simples à mettre en place et efficaces si l'application n'utilise pas de chiffrement SSL/TLS avec un HSTS (Strict Transport Security) bien configuré. La confusion entre les protocoles de chiffrement, le hachage et l'e ncodage peut créer des brèches, par exemple l'utilisation de fonctions base64 pour traiter des données sensibles au lieu d'un chiffrement. Selon l'OWASP, la prévalence et l'exploitabilité de ce risque sont faibles. En revanche, l'impact technique, quant à lui, est sévère, et c'est pour cette raison qu'il faut prendre en compte ce risque, surtout dans le cas où des exigences en termes de confidentialité et d'intégrité sont présentes.

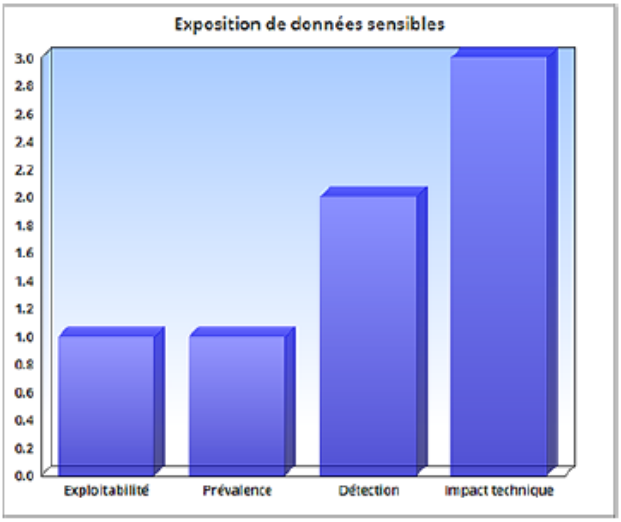


FIGURE 2.8 – Evaluation du risque de l'exposition de données sensibles

2.3.1.7 Manque de contrô le d'accès au niveau fonctionnel

Le manque de contrôle d'accès est un risque lié à des vulnérabilités retrouvées dans la gestion des droits d'une application, des défauts de conception ou l'utilisation de fonctions appropriées à l'intérieur du code. L'atta quant peut alors accéder à des fonctionnalités non autorisées, voire compromettre le site, avec des attaques du type déni de service.

FIGURE 2.9 – Evaluation du risque du manque de controle d'accès au niveau fonctionnel

2.3.1.8 Cross Site Request Forgery (CSRF)

Ces vulnérabilités permettent de forcer l'utilisateur à envoyer une requête HTTP à son insu afin de changer son mot de passe, acheter sur un site marchand ou le déconnecter d'une application. L'attaquant envoie un e-mail à un utilisateur avec un lien malicieux mais pouvant passer inaperçu. Ce lien renvoie vers un serveur hébergeant un script dont le but est d'envoyer un formulaire HTML au serveur web pour changer le mot de passe de l'utilisateur.

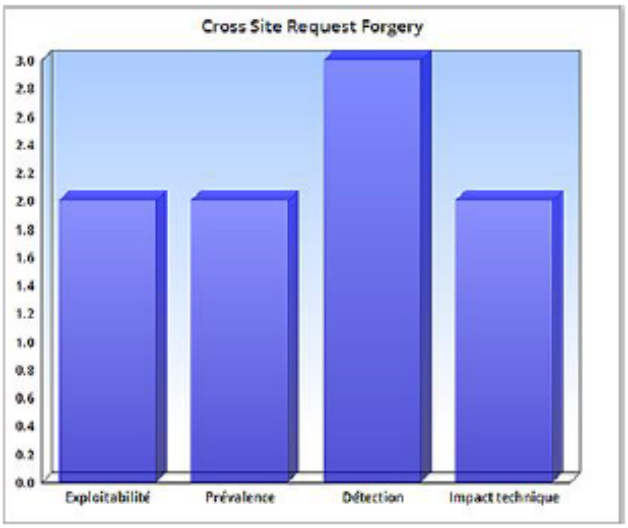


FIGURE 2.10 – Evaluation du risque du Cross Site Request Forgery

2.3.1.9 Exploitation de vulnérabilités connues

Le risque d'exploitation de vulnérabilités connues ne s'adresse pas seulement au périmètre du code mais à toute l'infrastructure d'une application comme les services web, les frameworks, les systèmes d'exploitation... Chaque année, de nouvelles vulnérabilités avec un impact sur des millions de systèmes sont dévoilées. Une des dernières fut Heartbleed, qui a impacté 17% des serveurs web dits sécurisés par le protocole TLS, dont Amazon Web Services, GitHub, Reddit, etc. Cette vulnérabilité autorise un cybercriminel à obtenir des informations en transit sur le serveur et ainsi voler les cookies, les sessions, les mots de passe des utilisateurs naviguant sur l'application. Heartbleed est simplement un exemple, chaque année des vulnérabilités de ce type sont divulguées ou pas. En effet, il existe des vulnérabilités dites « 0day », c'est à dire qu'elles ne sont pas connues, voire en vente sur le marché noir.

L'OWASP décrit le risque avec une prévalence très forte mais une détection faible car c'est souvent dans le domaine de la recherche que ce type de vulnérabilités est trouvé.

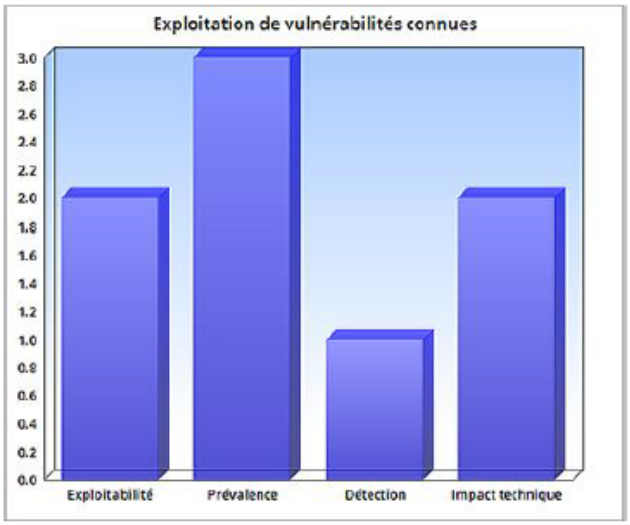


FIGURE 2.11 – Evaluation du risque de l'exploitation de vulnérabilités connues

2.3.1.10 Redirections et renvois non validés

Les redirections et les renvois non validés sont une vulnérabilité profitant d'une faiblesse dans le code et dont l'objectif est de rediriger l'utilisateur sur une page malveillante

ou administrative du site web. Il est courant de voir des redirections d'U RL interne sur la plupart des applications utilisant un modèle MVC (Model View Controller). Globalement bien configuré dans la plupart des frameworks, il peut s'avérer que certaines applications " from scratch " sont vulnérables et permettent à un attaquant de contourner les contrôles d'accès ou de rediriger vers une page de phishing suivant le cas.

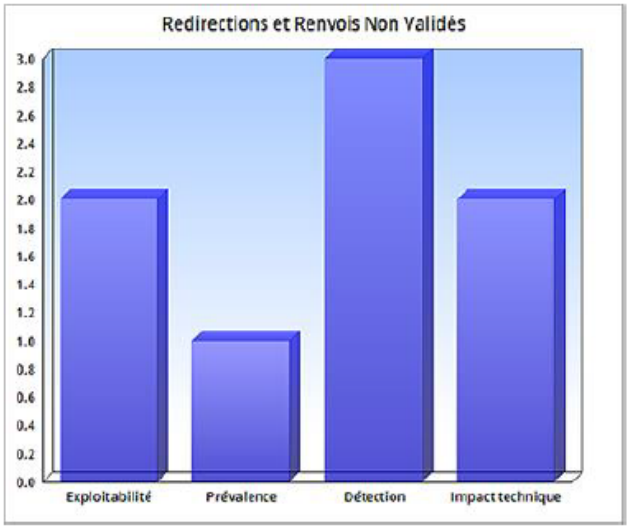


FIGURE 2.12 – Evaluation du risque des redirections et rencois non validés

Comme nous avons dit auparavant, pour prévoir les erreurs qu'un architecte ou un developpeur peut commettre, il faut tout d'abord comprendre les failles de sécurité les plus courantes (Voir la section 2.3.1) et ensuite se documenter sur les erreurs de développement et de conception. Ces erreurs sont listés dans une bibliothèque en ligne nommé CWE.

La bonne compréhension de ces erreurs nous aidera énormément dans la formulation des recommandations de sécurité (Une des mesures de notre méthodologie de développement sécurisé).

2.3.2 Les erreurs de développement et de conception "CWE"

Common Vulnerabilities and Exposures ou CVE est un dictionnaire des informations publiques relatives aux vulnérabilités de sécurité. Le dictionnaire est maintenu par l'organisme MITRE, soutenu par le département de la Sécurité intérieure des États-Unis.

Mieux vaut prévenir que guérir : C'est dans cette optique que nous avons decider de travailler avec la biliothèque CWE qui contient toutes les erreurs de développement et de conception connues. Une bonne compréhension de ses erreurs augmentera l'effectivité de la méthodologie que nous allons mettre en place par la suite.

2.3.2.1 Les erreurs de développement

L'erreur humaine occupe une position centrale dans les questions relatives à la sécurité dans le développement des applications. Heureusement que nous avons trouver CWE, qu'est une bibliothèque en ligne qui regroupe toutes les erreurs de développement.

NB : Nous pouvons réaliser des recommandations à base de ces erreurs, que le développeur doit suivre : "Mieux vaut prévenir que guérir"

- **Exemple :**

Voila un exemple d'une erreur que les développeurs commettent :

Si un utilisateur fait la saisie de la chaîne "twenty-one" pour val, l'entrée suivante est enregistrée :

- INFO : Failed to parse val=twenty-one

Si un utilisateur malveillant fait la saisie de la chaine "twenty-one%0a%0aINFO :+User+logged+out%3dbadouy" , l'entrée suivante sera enregistrée :
• INFO : Failed to parse val=twenty-one
• INFO : User logged out=badguy

FIGURE 2.13 – Les erreurs de développement "CWE"

Example Language: **Java** *(bad code)*

```
String val = request.getParameter("val");
try {

  int value = Integer.parseInt(val);
}
catch (NumberFormatException) {
  log.info("Failed to parse val = " + val);
}
...
```

FIGURE 2.14 – Exemple d'une erreur de développement

Il est clair que les attaquants peuvent utiliser ce même mécanisme pour **empoisonner les fichiers Log**.

2.3.2.2 Les erreurs de conception

Du point de vue de la sécurité, de nombreux experts considèrent l'architecture et la conception comme la phase la plus critique du SDLC. Les bonnes décisions prises au cours de cette phase non seulement produiront une approche et une structure plus résilientes et résistantes aux attaques, mais aideront également souvent à prescrire et à guider de bonnes décisions lors de phases ultérieures telles que le code et les tests. Les mauvaises décisions prises au cours de cette phase peuvent conduire à des défauts de conception

qui ne peuvent jamais être surmontés ou résolus même par le code et les tests les plus intelligents et les plus disciplinés.

FIGURE 2.15 – Les erreur de conception "CWE"

- **Exemple :**

Voila un exemple d'une erreur que les architectes commettent : "**CWE-223 : Omission of Security-relevant Information**"

Example Language: **PHP** *(bad code)*

```
function login($userName,$password){
  if(authenticate($userName,$password)){
    return True;
  }
  else{
    incrementLoginAttempts($userName);
    if(recentLoginAttempts($userName) > 5){
      writeLog("Failed login attempt by User: " . $userName . " at " +
      date('r') );
    }
  }
}
```

FIGURE 2.16 – Exemple d'une erreur dû à une mauvaise décision lors de la conception

Ce code enregistre uniquement les tentatives de connexion infructueuses lorsqu'une certaine limite est atteinte. Si un attaquant connaît cette limite, il peut empêcher la découverte de son attaque en évitant cette limite.

Avec la connaissance préalable des erreurs de conception et de développement, nous pouvons guider à la fois les architectes et les développeur pour minimiser les vulnérabilités de sécurité introduites dans ces deux phases.

Dans le chapitre suivant, nous allons integrer des mesures de sécurité dans le cycle de développement de logiciel SDLC. L'une des mesures de sécurité est la sensibilisation des architectes et développeurs. Nous allons réalisé des guides de sécurité qui vont contenir des recommandations. Elles seront basées sur les erreurs de conception et de développement (Voir section2.3.2).

Chapitre 3

Le cycle de développement de logiciel sécurisé S-SDLC

Après avoir étudié l'environnement de la sécurité web et les risques majeurs dans une application, nous allons pouvoir orchestrer tout cela à travers un cycle de développement sécurisé.

Introduction

La plupart des organisations adoptent une méthodologie de développement de systèmes (SDLC) pour le développement et la mise en œuvre de systèmes informatiques. Comme on a vu dans le chapitre précédent, SDLC est un processus de cycle de vie en plusieurs étapes destiné à fournir des systèmes informatiques garantissant des systèmes de qualité répondant aux spécifications et dans les délais et les coûts estimés.

Bien que la plupart des entreprises reconnaissent que la sécurité est un facteur important dans le développement de systèmes informatiques, les coûts et les performances de l'entreprise l'emportent souvent sur la sécurité. Bien que les problèmes de sécurité soient mieux connus, la plupart des entreprises se concentrent uniquement sur la sécurité dès la phase de mise en service du développement du système et tentent d'intégrer la sécurité à la conception finale, ce qui entraîne une application inefficace en terme de sécurité.

Un moyen efficace de protéger les systèmes informatiques contre les cyber-menaces consiste à **intégrer la sécurité à chaque étape du processus SDLC**, du lancement au développement, en passant par le déploiement et la mise au rebut éventuelle du système. Cela inclut l'incorporation des spécifications de sécurité dans la conception, une évaluation continue de la sécurité à chaque phase et le respect des meilleures pratiques.

3.1 S-SDLC

Un S-SDLC (Secure Software Development Life Cycle) a pour objectif d'intégrer des actions et des contrôles de sécurité dans un processus de développement des applications, le but étant d'améliorer la sécurité et de réduire les coûts de maintenance et de remise en marche d'une application après incident. Microsoft, qui a beaucoup contribué dans le domaine des cycles de développement sécurisés avec son process model Security Development Lifecycle (SDL), **estime qu'une vulnérabilité corrigée dans le code est en moyenne cent fois moins chère après incident**.

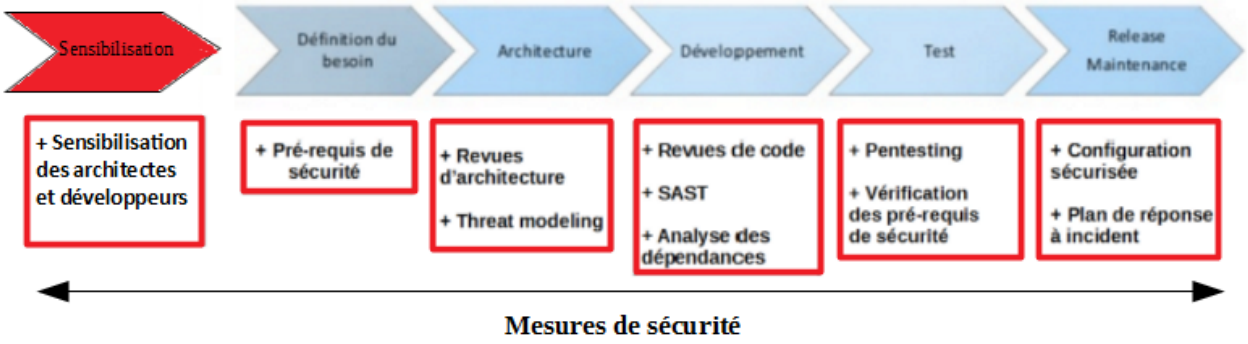


FIGURE 3.1 – Le cycle SDLC sécurisé

Nous allons commencé par la première mesure de sécurité qu'est la sensibilisation.

3.1.1 Sensibilisation

Tous les membres des équipes de développement doivent recevoir une formation appropriée pour être informés des bases de la sécurité et des évolutions récentes dans les domaines de la sécurité et du respect de la vie privée. Les personnes qui jouent un rôle technique (développeurs, testeurs et chefs de projet) et qui sont directement impliquées dans le développement de logiciels, doivent participer au moins à une formation sur la sécurité chaque année.

3.1.1.1 Sensibilisation des architectes

L'un des défis majeurs que nous pouvons rencontrer lors de l'intégration de la sécurité dans la conception de nos projets logiciels est la pénurie d'architectes expérimentés qui ont une solide compréhension des problèmes de sécurité.

Pour résoudre ce problème, nous avons élaborer le « **guide de l'architecte** », une liste d'énoncés décrivant des bonnes pratique de sécurité requises par les architectes. Ce guide joue un rôle clé dans la sensibilisation de l'importance cruciale que joue la sécurité dans la phase de conception dans le cycle de développement de logiciel. Il nous garantira la réduction du nombre de défauts le plus tôt possible, tout en minimisant les ambiguïtés

et autres faiblesses.

Les ressources de connaissances (telles que les principes de sécurité, les consignes de sécurité et les modèles d'attaque) constituent le carburant qui alimente efficacement le processus d'analyse des risques architecturaux. Ils vont vous guidés en suggérant les questions à poser et les mesures d'atténuation à prendre.

Principes de sécurité :

Bien que les projets doivent toujours faire appel à au moins un architecte de sécurité logicielle véritablement expérimenté, des ressources de connaissance telles que **les principes de sécurité**, les consignes de sécurité et les modèles d'attaque peuvent aider les entreprises à répartir plus efficacement ces ressources limitées entre projets.

Les principes de sécurité sont un ensemble de pratiques de haut niveau issues d'une expérience réelle qui peuvent aider les architectes à créer des logiciels plus sûrs.

Ces principes définissent des pratiques efficaces qui s'appliquent principalement aux décisions logicielles au niveau architectural et qui sont recommandées quelles que soient la plate-forme ou le langage du développement. Les principes de sécurité cités dans le guide de l'architecte sont :

1. Principe du moindre privilège.
2. Principe de l'échec sécurisé.
3. Principe de sécurisation du maillon le plus faible.
4. Principe de défense en profondeur.
5. Principe de la séparation des privilèges.
6. Principe d'économie de mécanisme.
7. Principe de « Ne jamais supposer que vos secrets sont en en sécurité ».
8. Principe de la médiation complète.
9. Principe de l'acceptabilité psychologique.
10. Principe de la conception ouverte.

1. Principe du moindre privilège

L'octroi d'autorisations à un utilisateur au-delà des droits nécessaires peut permettre à cet utilisateur d'obtenir ou de modifier des informations de manière non souhaitée.

« une délégation prudente des droits d'accès peut limiter la capacité des attaquants à endommager un système. »

2. Principe de l'échec sécurisé

Lorsqu'un système tombe en panne, il devrait le faire en toute sécurité. Ce comportement comprend généralement plusieurs éléments:

- valeurs par défaut sécurisées (la valeur par défaut consiste à refuser l'accès)
- Annulation des modifications et restauration de système dans un état sécurisé.

La confidentialité et l'intégrité d'un système ne doivent pas être altérées, même si la disponibilité a été perdue.

En cas de panne,

- Les attaquants ne doivent pas obtenir, en aucun cas, des droits d'accès aux objets privilégiés qui sont normalement inaccessibles.
- Un système révélant des informations sensibles sur la panne à des attaquants potentiels pourrait fournir des informations supplémentaires que les acteurs des menaces pourraient ensuite utiliser pour créer une attaque ultérieure.

« Déterminez ce qui peut se produire lorsqu'un système tombe en panne et assurez-vous que cela ne le menace pas. »

FIGURE 3.2 – Les principes de sécurité "Guide de l'architecte"

Recommandations de sécurité :

« **Secure by Design** » est une approche permettant de développer des systèmes logiciels sécurisés. Elle déplace l'assurance logicielle de la recherche des failles de sécurité à l'identification des défauts d'architecture et de conception **[W10]**.

Pour satisfaire aux exigences de sécurité, les architectes de logiciels doivent adoptés des tactiques de sécurité. Ces tactiques architecturales fournissent des mécanismes pour **résister, détecter, réagir après une attaque**.

Les trois catégories sont importantes : En utilisant une analogie familière, verrouiller votre porte est une forme de résistance à une attaque, avoir un détecteur de mouvement à l'intérieur de votre maison est une forme de détection d'une attaque et avoir une assurance est une forme de rétablissement après une attaque.

La figure ci dessous montre les objectifs des tactiques de sécurité.

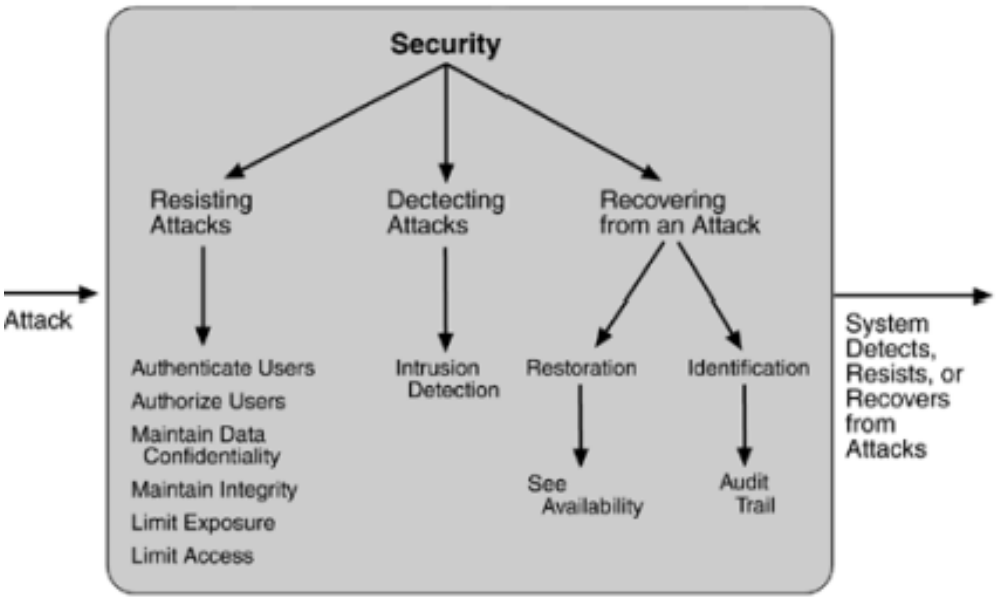


FIGURE 3.3 – Les tactiques de sécurité "Guide de l'architecte"

Néanmoins, des failles dans la mise en œuvre des tactiques de sécurité ou leur détérioration au cours de l'évolution et de la maintenance des logiciels peuvent introduire de graves vulnérabilités qui pourraient être exploitées par des attaquants. Dans cette section,

nous présentons le concept de CAWE (Common Architectural Weakness Enumeration), qui énumère les types de vulnérabilités courants enracinés dans l'architecture d'un logiciel et fournit des techniques d'atténuation permettant de les résoudre. Le catalogue CAWE contient 224 faiblesses liées à la phase de conception du cycle de développement de logiciel.

À travers ce catalogue, nous visons à promouvoir la prise de conscience des failles de sécurité au niveau de la phase conception.

3	**Gestion de privilège** Le **privilège** est défini comme la délégation d'autorité sur un système informatique. Un privilège permet à un utilisateur d'effectuer une action. Parmi les différents privilèges, citons la possibilité de créer un fichier dans un dossier, de lire ou de supprimer un fichier, d'accéder à un périphérique ou d'obtenir des autorisations de lecture ou d'écriture sur un socket pour la communication par Internet.
OWASP Top 10 and CWE/SANS Top 25 Elements	**Recommandations**
	R1. Compartimentez le système pour avoir des zones "sûres" où des limites de confiance peuvent être tracées sans ambiguïté. Ne laissez pas les données sensibles sortir de la limite de confiance et soyez toujours prudent lorsque vous vous connectez à un compartiment en dehors de la zone sécurisée.
	R2. Les bibliothèques chargées doivent être bien comprises et provenir d'une source fiable. L'application peut exécuter du code contenu dans les bibliothèques natives, qui contiennent souvent des appels susceptibles de poser d'autres problèmes de sécurité, tels que des dépassements de mémoire tampon ou l'injection de commandes. Toutes les bibliothèques natives doivent être validées pour déterminer si l'application nécessite l'utilisation de la bibliothèque. Il est très difficile de déterminer ce que font réellement ces bibliothèques natives et le potentiel de code malveillant est élevé.
	R3. Exécutez votre code en utilisant les privilèges les plus bas nécessaires pour accomplir les tâches nécessaires. Si possible, créez des comptes isolés avec des privilèges limités qui ne sont utilisés que pour une tâche unique. De cette manière, une attaque

FIGURE 3.4 – Extrait des recommandations du guide de l'architecte

3.1.1.2 Sensibilisation des développeurs

Cette section liste les meilleures pratiques permettant de sensibiliser et aider les équipes de développement à créer des applications plus sécurisées. C'est une première étape vers la construction d'une base de connaissances autour de la sécurité des applications web. Cette liste permet d'identifier les règles minimales nécessaires pour neutraliser les failles dans les applications les plus critiques.

Nous avons réaliser un guide pour les développeur. Ce guide contient des recomman-

dations concernant :

- L'authentification
- La gestion des sessions
- La protection des données (locales et en transit)
- La gestion des entrées et sorties
- Le contrôle d'accès et la gestion des erreurs
- La journalisation

6	**Gestion des erreurs** La gestion des erreurs garantit qu'en cas d'erreur, aucune information importante n'est présentée à l'utilisateur. Il est à noter, que lors de la phase de reconnaissance, un attaquant stresse l'application pour recueillir le maximum d'information à travers les messages d'erreurs. Pour s'y protéger, ci-dessous les recommandations à suivre :
OWASP Top 10 and CWE/SANS Top 25 Elements	**Recommandations**
	R1. Afficher des messages d'erreurs génériques : Les messages d'erreurs ne doivent pas révéler des détails sur l'état interne de l'application. Par exemple, il ne faut jamais exposés à l'utilisateur via des messages d'erreurs le chemin du système de fichiers ou la pile d'informations.
	R2. Gérer toutes les exceptions : Les gestionnaires d'erreurs doivent être configurés pour gérer les erreurs inattendues et contrôler minutieuse- ment toutes les sorties possibles.
	R3. Gérer les erreurs générées par les Framework : la plateforme de développement peut générer des messages d'erreurs par défaut qui révèlent parfois des informations importantes. Ces messages devraient être remplacés par des messages d'erreurs personnalisés.

FIGURE 3.5 – Extrait des recommandations du guide du développeur

Pour aider les developpeurs à bien assimiler les recommandations cités dans le guide du développeur, nous avons créer **un quizz** contenant des questions à choix multiple.

Notre quizz est accessible depuis le site-web suivant :

https ://forms.office.com/Pages/ResponsePage.aspx ?id=DQSIkWdsW0yxEjajBLZtrQAAAAAAAAAAAMAAPzg63tUQjU1OFFYWTVOUDRFWk9NT1pFUTZUTTJSVi4

Ci-dessous un extrait de notre quizz :

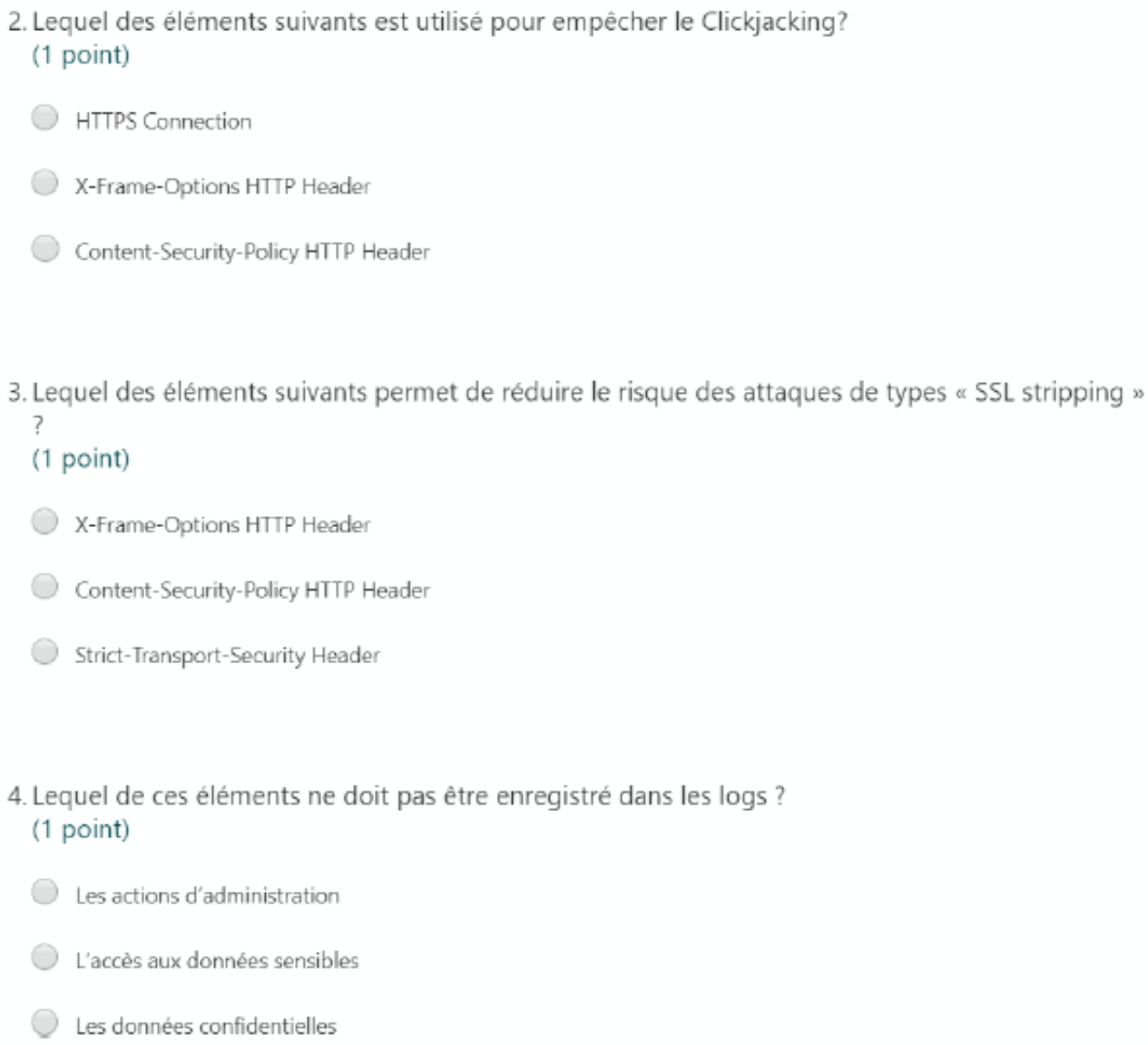


FIGURE 3.6 – Extrait du Quizz du développeur

Les concepteurs et développeurs devraient toujours appliquer les principes et recommandations cités dans les deux guides. Cela aidera d'une part, à réduire le nombre de failles de sécurité introduites dans leurs architectures et codes, et d'autre part à minimiser le temps d'execution du cycle SDLC.

3.1.2 Intégration des bonnes pratiques

Réussir à maintenire la sécurité lors de la conception et de la mise en oeuvre des applications est un défi que nombreuses organisations prennent à la légère. Il est pourtant reconnu que la vulnérabilité des applications logicielles a eu des conséquences majeures sur la qualité des niveaux de sécurité des entreprises au cours des dernières années .

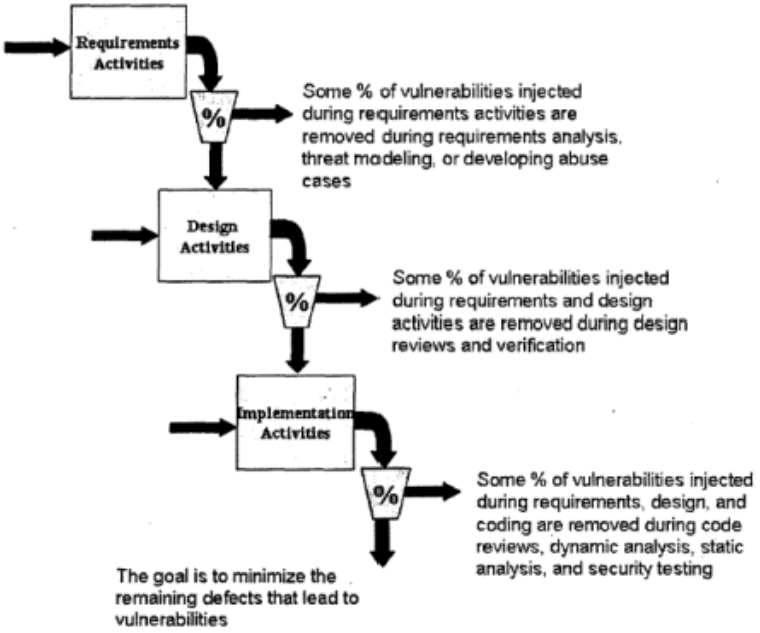


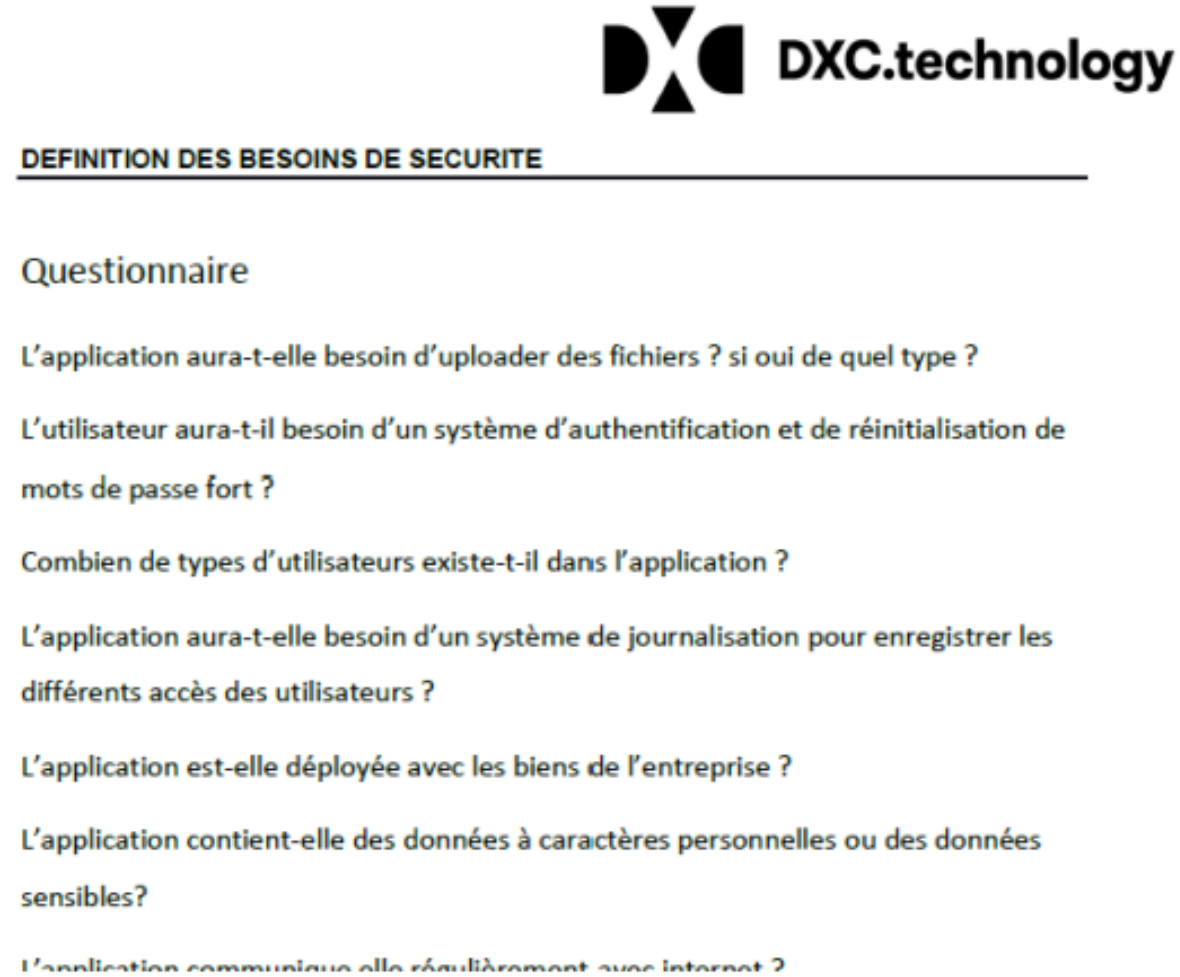
FIGURE 3.7 – Integration des mesures de sécurité dans le cycle SDLC **[W11]**

une intégration adéquate des mesures de sécurité dans l'ensemble du cycle de vie des applications est nécessaire.

3.1.2.1 Phase « Définition du besoin»

La nécessité de prendre en compte, dès le début du projet, la sécurité et le respect de la vie privée constitue un principe de base du développement sécurisé. Le meilleur moment pour définir les exigences en sécurité pour un logiciel se situe au début des étapes initiales de planification. Ceci étant dit, il est nécessaire de définir les exigences de sécurité dans cette phase. Pour cela nous avons conçu un document « **Definition des besoin de sécurité** ». Ce document est composé d'un questionnaire et un tableau.

Le questionnaire :



DXC.technology

DEFINITION DES BESOINS DE SECURITE

Questionnaire

L'application aura-t-elle besoin d'uploader des fichiers ? si oui de quel type ?

L'utilisateur aura-t-il besoin d'un système d'authentification et de réinitialisation de mots de passe fort ?

Combien de types d'utilisateurs existe-t-il dans l'application ?

L'application aura-t-elle besoin d'un système de journalisation pour enregistrer les différents accès des utilisateurs ?

L'application est-elle déployée avec les biens de l'entreprise ?

L'application contient-elle des données à caractères personnelles ou des données sensibles?

L'application communique elle régulièrement avec internet ?

FIGURE 3.8 – Extrait du questionnaire "Définition du besoin"

Le questionnaire contient un ensemble de questions générales visant à identifier les exigences de sécurité du client , ces questions seront destinés au client, et donc elles sont simples et ne contient pas des mots technique. Nos questions visent essentiellement les axes suivants :

- Gestion des entrées et sorties.
- Authentification et gestion de session.
- Contrôle d'accès.
- Gestion d'erreur.
- Journalisation.

- Connexions aux systèmes externes.

« **L'application aura t'elle besoin d'uploader des fichiers ? si oui de quel type ?**» Avec cette question on peut définir le niveau de sécurité que nous devons imposer en ce qui concernce la validations des entrées. Le niveau sécurité est mineur pour les fichiers texte, modéré pour les fichiers images, mais doit impérativement être majeur pour les fichiers exécutables.

Le tableau :

		Sécurité		
		Majeur	Modéré	Mineur
Gestion des entrées et sorties	Validation d'entrée			
	Encodage sortie			
Authentification et gestion de session	Authentification			
	Gestion de session			
Contrôle d'accès	Contrôle d'accès			
Gestion d'erreur	Gestion des erreurs			
Journalisation	Journalisation			
Connexions aux systèmes externes	Protection des données en transit			
Chiffrement	Protection des données locales			
CIA	Confidentialité			

FIGURE 3.9 – Extrait du tableau "Définition du besoin"

Si notre client nous affirmes que l'application aura besoin d'uploader des fichiers de type executable, nous devons coché la case "Sécurité Majeur" pour la section "Validation d'entrée". Cette charte est toujours visionner par l'equipe SOC dans toutes les phases du cycle de développement pour veiller au respect des exigences de sécurité.

Une "Sécurité Majeur" dans la section "Validation d'entrée, se traduit par des exigences que les concepteurs et developpeurs doivent remplir. Ces exigences sont regroupés dans le document « **Check list de sécurité** », et sont verifier par l'équipe durant dans chaque phase.

Recommandations	Check
1. Toutes les validations de données se font sur un système approuvé.	
2. Les sources de données sont identifiés et classées en "sources fiables" et "sources non fiables".	
3. Les données provenant de sources non fiables sont examiné et validé.	
4. L'existence d'une routine de validation d'entrée et le processus de validation est centralisé.	
5. L'encodage de caractère est spécifie pour toutes les sources d'entrée (UTF 8 ...)	
6. L'encodage des données est effectué à l'aide d'un encodage de caractères commun (Avant la validation)	
7. Les échecs de validation entraînent automatiquement le rejet des entrées.	
8. Les données fournies par le client sont validées avant le traitement, y compris tous les paramètres, les URL et le contenu de l'en-tête HTTP.	
9. Les valeurs d'en-tête des demandes et des réponses ne contiennent que des caractères ASCII.	
10. Les données des redirections sont validés (un attaquant peut soumettre du contenu malveillant directement à la cible de la redirection, contournant ainsi la logique de l'application et toute validation effectuée avant la redirection)	
11. Les types de données attendus sont validés.	
12. La longueur des données est validé.	
13. Une liste «blanche» de caractères autorisés est réalisé.	
TOTAL « VALIDATION DES ENTRÉES »	.../13

FIGURE 3.10 – Extrait de la Check-list 'Section validation des entrée'

AXE	SCORE
Validation des entrées	....
Encodage de sortie	....
Authentification et gestion de mots de passe	....
Gestion des sessions	....
Contrôles d'accès	....
Journalisation	....
Protection de données	....
Sécurité de la base de données	....
Gestion de mémoire	....
TOTAL	..../102

FIGURE 3.11 – Bilan de la Check-list

La figure ci-dessous illustre les directives de sécurité suivies dans la phase de définition des besoins.

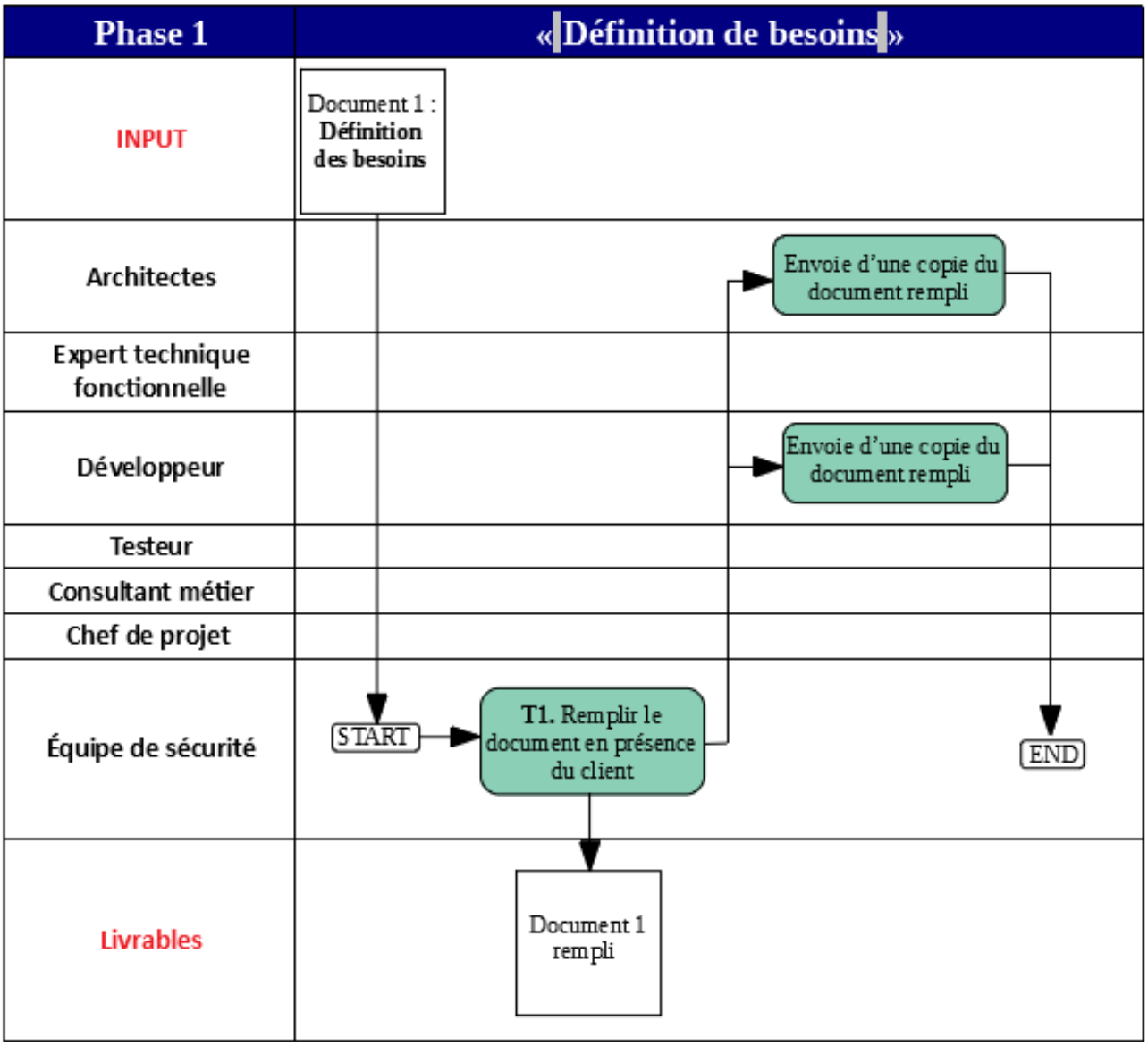


FIGURE 3.12 – Processus "Definition de besoin"

3.1.2.2 Phase « Architecture»

L'architecture et la conception logicielles sont l'endroit où les ambiguïtés et les idées sont traduites et transformées en réalité.

Du point de vue de la sécurité, de nombreux experts considèrent l'architecture et la conception comme la phase la plus critique du SDLC. Les bonnes décisions prises au cours de cette phase non seulement produiront une approche et une structure plus résilientes et résistantes aux attaques, mais aideront également souvent à prescrire et à guider de bonnes décisions lors de phases ultérieures telles que le code et les tests. Les mauvaises décisions prises au cours de cette phase peuvent conduire à des défauts de conception qui ne peuvent jamais être surmontés ou résolus même par le code et les tests les plus intelligents et les plus disciplinés.

« Aujourd'hui, si la sécurité logicielle porte essentiellement sur la saturation de mémoire tampon, l'injection SQL et d'autres problèmes d'implémentation, la réalité est qu'environ la moitié des défauts de sécurité détectés dans les logiciels actuels sont en réalité imputables à des défauts d'architecture et de conception » **[McGraw 2006]**

L'architecture doit traiter non seulement les problèmes de sécurité actuellement connus, à la fois les faiblesses et les attaques connues, mais aussi, à son niveau d'abstraction, doit être faible et résiliente en prenant en compte les critères de sécurité en constante évolution.

Modélisation des menaces

La modélisation des menaces est employée dans des environnements où il existe un risque sérieux de sécurité. L'équipe de développement prend en compte, documente et étudie les implications de la conception dans le domaine de la sécurité, dans le contexte de l'environnement opérationnel prévu, et de façon structurée. La modélisation des menaces permet aussi de prendre en considération des problèmes de sécurité au niveau de l'application ou d'un composant. C'est un travail d'équipe impliquant les chefs de projet, les développeurs et les testeurs. La modélisation des menaces représente la principale tâche d'analyse de la sécurité effectuée au cours de la phase de conception du logiciel.

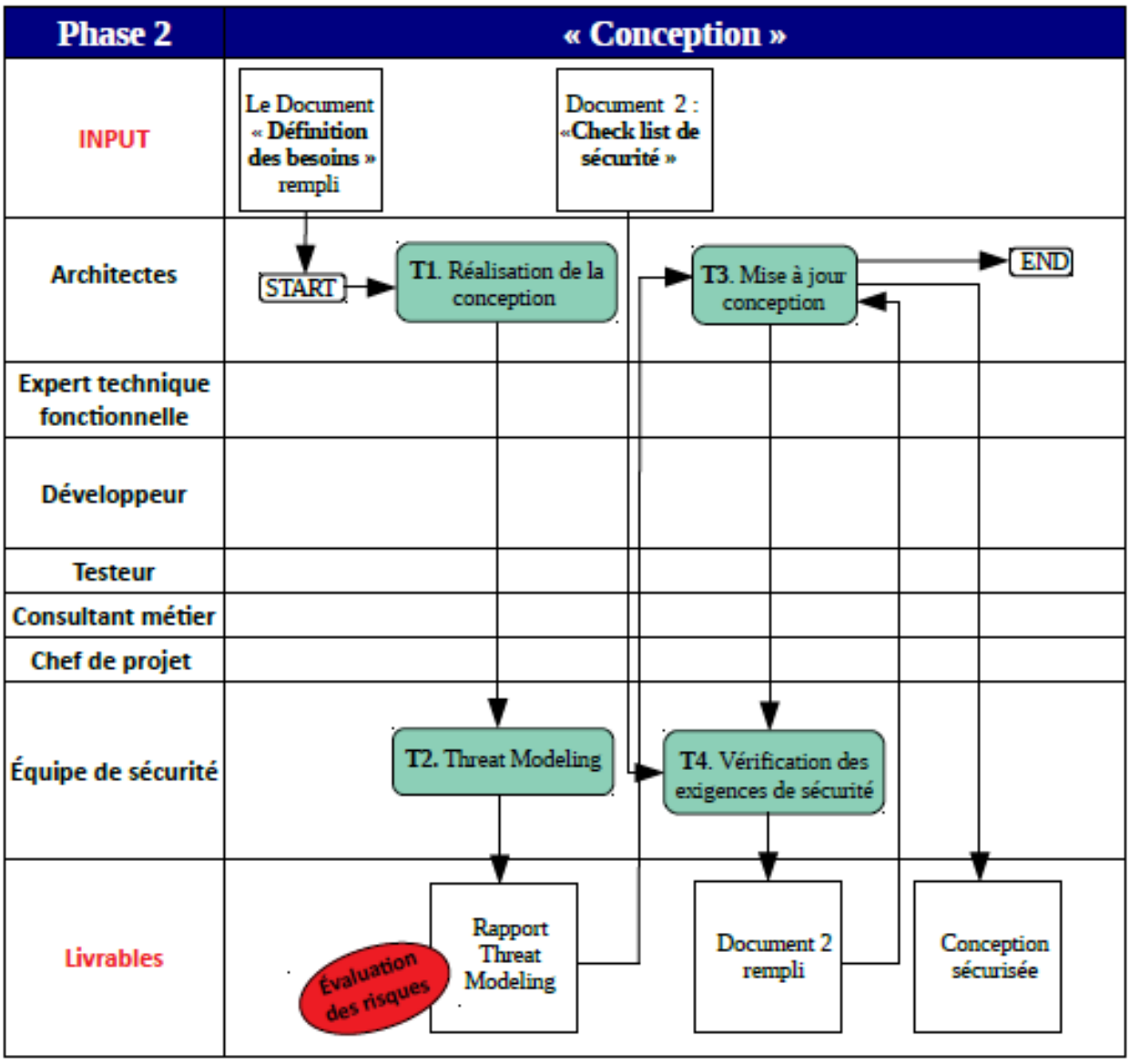


FIGURE 3.13 – Processus "Conception"

3.1.2.3 Phase « Developpement»

Au cours de cette phase, l'équipe de développement doit tester le logiciel, minutieusement, du point de vue de la sécurité. L'objectif est de prévenir les failles de sécurité et de corriger celles qui pourraient survenir. Cette action minimise le risque de retrouver ces failles dans la version finale du logiciel.

L'analyse statique est une méthode permettant d'inspecter le code source, le langage intermédiaire compilé ou le composant binaire à la recherche de défauts. Il recherche les problématiques connus basés simplement sur la logique de l'application, plutôt que sur le comportement de l'application pendant son exécution. La logique SAST peut être aussi simple qu'une expression régulière recherchant une API interdite via une recherche de texte, ou aussi complexe qu'une logique de flux de données qui semble permettre à des données altérées d'attaquer l'application.

SAST a tendance à fournir une très bonne couverture et, avec des solutions allant des outils gratuits ou open source aux offres commerciales complètes, plusieurs optons sont disponibles pour une organisation, quel que soit son budget. Cependant, tous les problèmes de sécurité ne se prêtent pas à la découverte via une analyse de modèle ou de flux de contrôle (par exemple, des vulnérabilités dans la logique métier). Les outils d'analyse statique sur un site peuvent faire appel à des experts humains pour ajuster la configuration de l'outil et trier les résultats afin de rendre les résultats plus exploitables pour les développeurs et de réduire les taux de faux positifs.

SAST	License	Version	Description
FindBugs	Free	3.01	Supports any JVM language and can detect 113 different vulnerability types.
Fortify SCA	Commercial	4.42	Supports 23 programming languages and detects over 700 vulnerabilities.
Jlint	Free	3.1.2	Works only with Java language. It helps to find more than 50 semantic and syntactic bugs.
OWASP LAPSE+	Free	2.8.1	Works only with Java language. The tool can identify 12 vulnerability types.
OWASP YASCA	Free	2.2	Supports 14 programming languages and aggregates results from 11 static analysis tools.
PMD	Free	5.5.1	Supports 20 programming languages and facilitates finding more than 25 bug types.
SonarQube	Free	5.6	Supports 20 programming languages and covers OWASP Top 10 vulnerability types.

FIGURE 3.14 – Comparaison des outils d'analyse statique

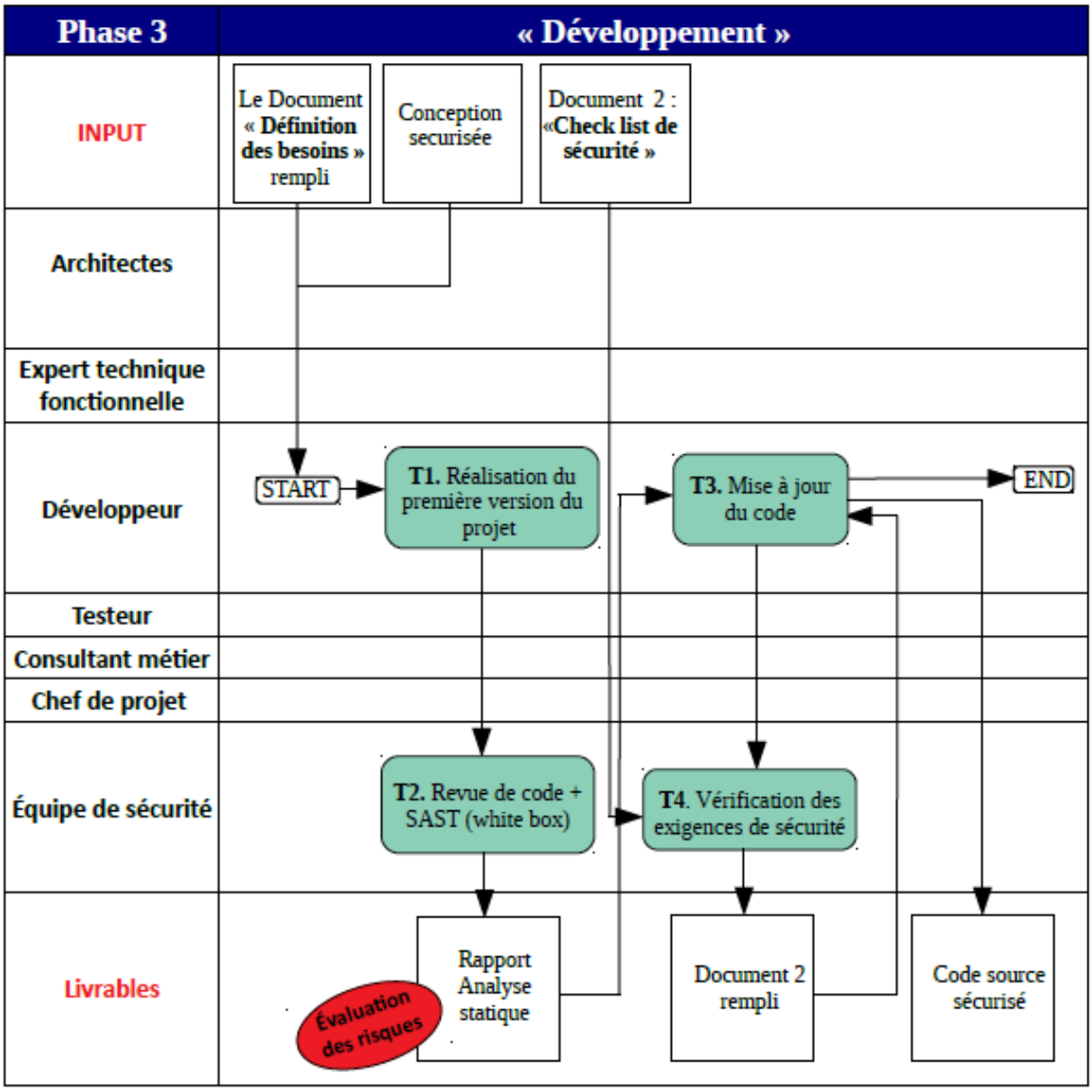


FIGURE 3.15 – Processus "Developpement"

3.1.2.4 Phase « Test»

À ce stade du développement, l'ensemble des fonctionnalités ont été intégrées au logiciel. Avant sa publication, l'application est également vérifié à l'aide d'outils d'analyse dynamique qui signalent également les défaut.

DAST

Un outil DAST (Dynamic Application Security Testing) est un programme qui communique avec une application Web par le biais de l'interface Web afin d'identifier les vulnérabilités de sécurité potentielles de l'application Web et les faiblesses architecturales. Il effectue un test de boîte noire . Contrairement aux outils de test de sécurité d'application statique, les outils DAST n'ont pas accès au code source et détectent donc les vulnérabilités en effectuant des attaques. Les outils DAST permettent de faire des analyses sophistiquées, détectant les vulnérabilités avec un minimum d'interactions utilisateur, une fois configuré avec le nom d'hôte, les paramètres d'analyse et les informations d'authentification. Ces outils tenteront de détecter les vulnérabilités dans les chaînes de requête, les en-têtes, les fragments ...

WIVET (Web Input Vector Extractor Teaser) est un projet qui calcul dans quelle mesure un scanner est capable d'analyser une application et de localiser facilement les vecteurs d'entrée en présentant une collection de challengers contenant des liens, des paramètres et des méthodes de livraison d'entrée que le processus d'analyse devrait localiser et extraire.

Rank	#	Logo	Vulnerability Scanner	Version	Vendor	WIVET Score	Chart
1			arachni	1.1	Tasos Laskos	96.00% Detection Rate	
2			Acunetix WVS Free Edition	8.0	Acunetix	92.00% Detection Rate	
3			Netsparker Community Edition	3.1.6.0	Netsparker Ltd	91.00% Detection Rate	
4			ZAP	2.2.2	OWASP	73.00% Detection Rate	
5			Vega	1.0	Subgraph	50.00% Detection Rate	

FIGURE 3.16 – Comparaison des outils d'analyse dynamique

Dans la partie simulation, nous avons utilisé l'outil Acunetix.

Fuzzing

Le fuzzing est une forme spécialisée de DAST (ou dans certains cas limités, SAST). Le fuzzing consiste à générer ou à muter des données et à les transmetre aux analyseurs de données d'une application (analyseur de fichiers, analyseur de protocole réseau, analyseur de communicaton interprocessus, etc.) pour voir leur réaction. Bien qu'il soit improbable qu'un exemple d'entrée fuzzée découvre un comportement vulnérable, le processus sera répété des milliers, voire des millions de fois (en foncton de la profondeur avec laquelle le logiciel a été fuzzé). L'avantage du fuzzing est qu'une fois l'automatisation configurée, elle peut exécuter d'innombrables tests distincts sur le code, le seul coût étant le temps de calcul. Le fuzzing produit une meilleure couverture du code d'analyse que le DAST ou le SAST traditionnel. Toutefois, selon le scénario, la configuration peut être laborieuse. De plus, le débogage des problèmes détectés par le fuzzing prend souvent beaucoup de temps. Le fuzzing aura le meilleur résultat en termes d'analyseurs syntaxiques écrits dans des langages non sécurisés pour la mémoire (par exemple, C, C ++), auxquels un attaquant peut accéder facilement avec des données malveillantes. Un modèle de menace est très utile pour identifier ces analyseurs.

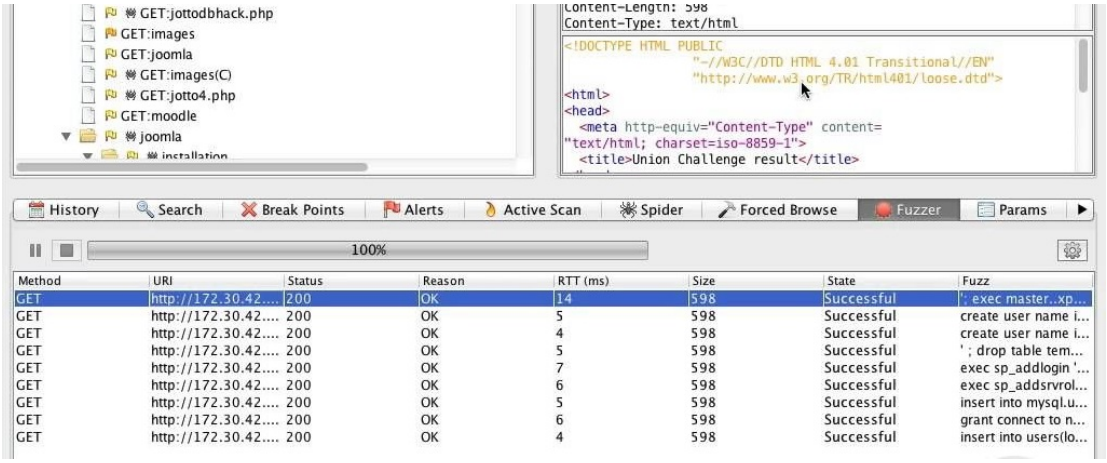


FIGURE 3.17 – Fuzzing avec l'outil OWASP ZAP

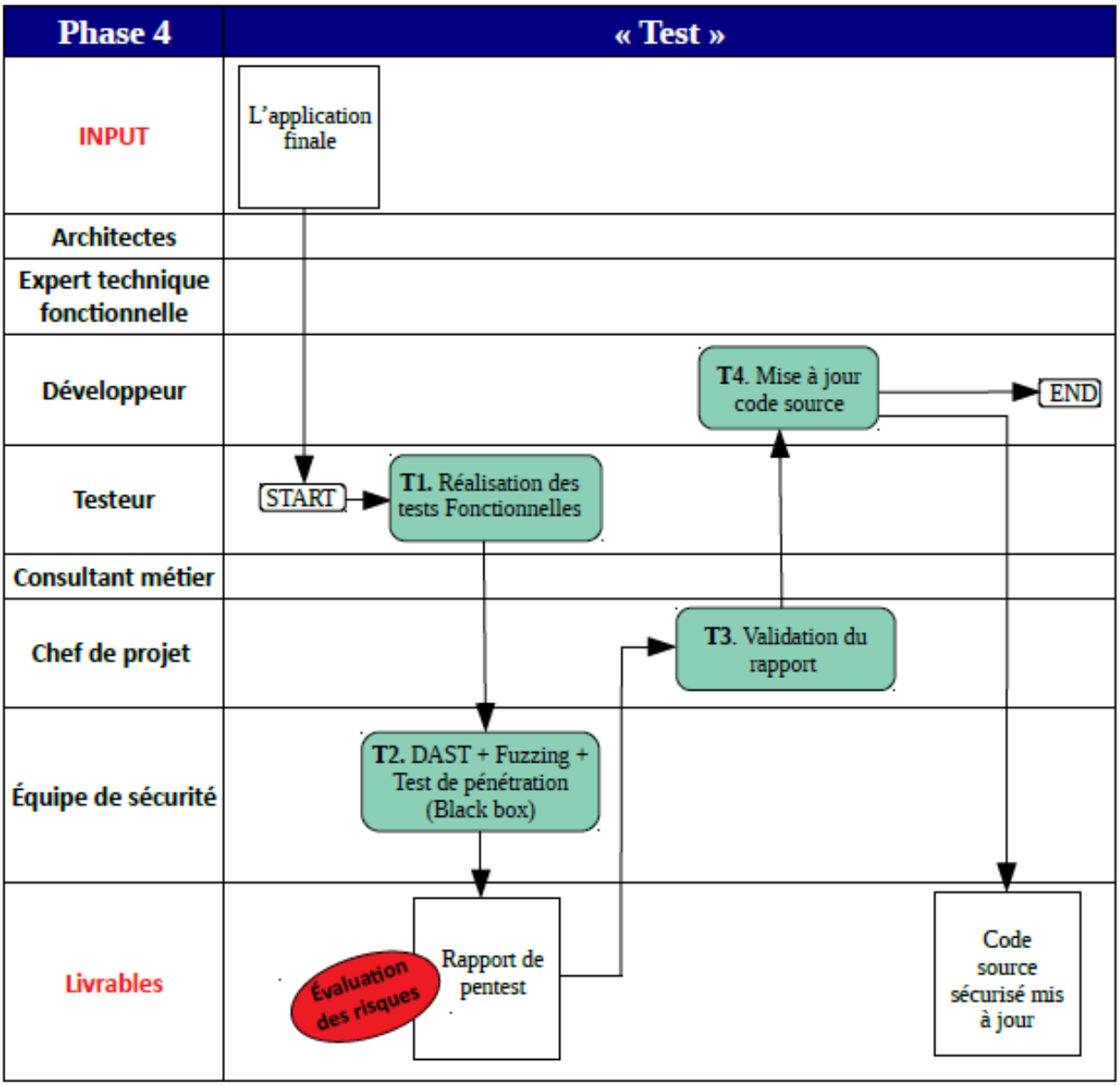


FIGURE 3.18 – Processus "Test"

Pour la méthodologie de Pentesting, nous avons choisi celle d'OWASP.

Malheureusement, pour la phase de déploiement, nous avons pa pu integrer des mesures de sécurité par manque de temps.

Durant chaque test réalisé par l'équipe de sécurité, des failles de sécurité remontent à la surface. Ces failles doivent etre évaluer et lister selon leur criticité. Plusieurs méthodologies propose une méthode d'évaluation (ie : L'attribution d'un score) de ces failles. La section suivante décrit la méthodologie d'OWASP.

3.2 Gestion de risque

Dans le domaine de la gestion de risque, les spécialistes définissent le risque comme la combinaison d'un impact et de la probabilité d'une menace. La menace est en l'occurrence un événement qui peut empêcher l'entreprise ou l'organisation d'atteindre ses objectifs.

Le graphique suivant illustre le processus de gestion de risque selon la norme ISO 31 000 : 2009.

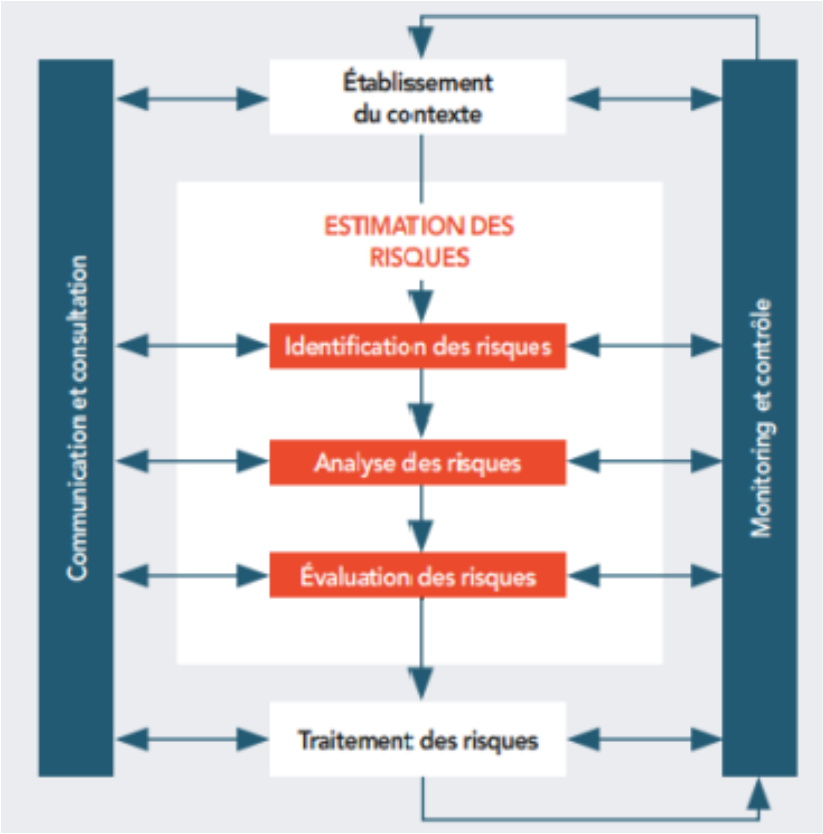


FIGURE 3.19 – Processus de gestion de risque **[W12]**

• Analyse des risques : comprendre la nature, les sources, les causes et les conséquences du risque.

• Évaluation des risques : déterminer si un risque ou un niveau de risque relatif à une vulnérabilité est acceptable ou tolérable.

L'OWASP propose la méthodologie "Risk Rating Methodology". Celle-ci est utilisée principalement dans les tests manuels de sécurité où les consultants interagissent directement avec le système d'information de l'entreprise.

Cette méthodologie permet d'estimer le risque associé à une vulnérabilité. Commençons par le modèle de risque standard proposé : **RISQUE = IMPACT * PROBABILITE** L'équation ci-dessous permet d'estimer la valeur de la probabilité et l'impact mentionnés dans le paragraphe précédent. Par la suite, on peut associer à chaque composant du risque un niveau de gravité. $P = \frac{S}{N} et I = \frac{S}{N}$ Où P et I désignent respectivement la probabilité et l'impact. S représente la somme des valeurs des facteurs. N représente le nombre des facteurs.

Probabilité et Niveau d'impact	
De 0 à <3	Faible
De 3 à <6	Moyen
De 6 à 9	Elevé

FIGURE 3.20 – Probabilité et niveau d'impact **[W6]**

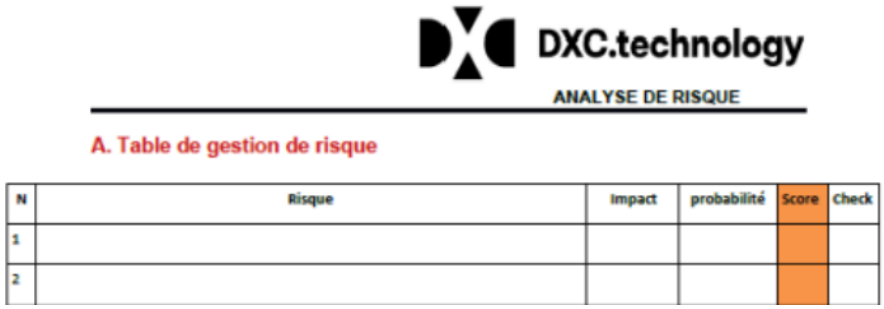
La phase finale de cette méthodologie consiste à estimer, selon le tableau ci-dessous, un niveau de gravité globale du risque.

	Estimation de gravité			
Impact	Elevé	Moyen	Elevé	Critique
	Moyen	Faible	Moyen	Elevé
	Faible		Faible	Moyen
		Faible	Moyen	Elevé
	Probabilité			

FIGURE 3.21 – Estimation de gravité **[W6]**

Les nouveaux risques qui ont remontés à la surface durant une phase du cycle du développement (Conception : Threat Modeling | Developpement : SAST .. | Testing : Pentesting ..) sont listés selon leurs score.

On essaye de trouver des solutions aux risques qui ont un score elevé. Pour les risque résiduels, on réalise un plan de réponse incident.


DXC.technology

ANALYSE DE RISQUE

A. Table de gestion de risque

N	Risque	Impact	probabilité	Score	Check
1					
2					

FIGURE 3.22 – Table de gestion de risque

Après avoir vu notre méthodologie de développement sécurisée, dans la chapitre suivant nous allons réalisé des simulations pour mesurer l'effectivité de cette méthodologie. Les tests sont réalisés sur des applications réelles.

Chapitre 4

Simulation

Le présent chapitre présente des simulations réelles réalisées afin de mesurer l'effectivité de notre méthodologie. Nous allons réaliser un audit de code source 'Phase développment', en plus d'un Pentest de site web 'Phase Test'.

Introduction

Une simulation réelle nous aidera, surement, à savoir les points forts et faible de notre méthodologie. Normalement la simulation de la solution doit etre réaliser depuis le tout début du cycle du développement (ie : La phase de définition des besoins), sauf que la durée du cycle peut s'avérer très longue. En tant que stagiaires limité en temps, nous avons été parvenu à integrer les mesures de sécurité à des projets développés, ou en cours de développement.

Cette simulation se basera sur un scan de vulnérabilités d'un site web (Phase Test) et une analyse statique du code (Phase de développement) .

Par la suite, les noms des projets seront masqués pour des raisons de confidentialité.

4.1 Analyse dynamique d'un site web

Durant cet analyse de type **Black Box**, on mettera le point, seulement, sur les failles de risque moyen et majeure.

Voici un exemple d'une vulnérabilité de niveau moyen qui a été trouvée à l'aide de l'outil Acunetix et OWASP ZAP :

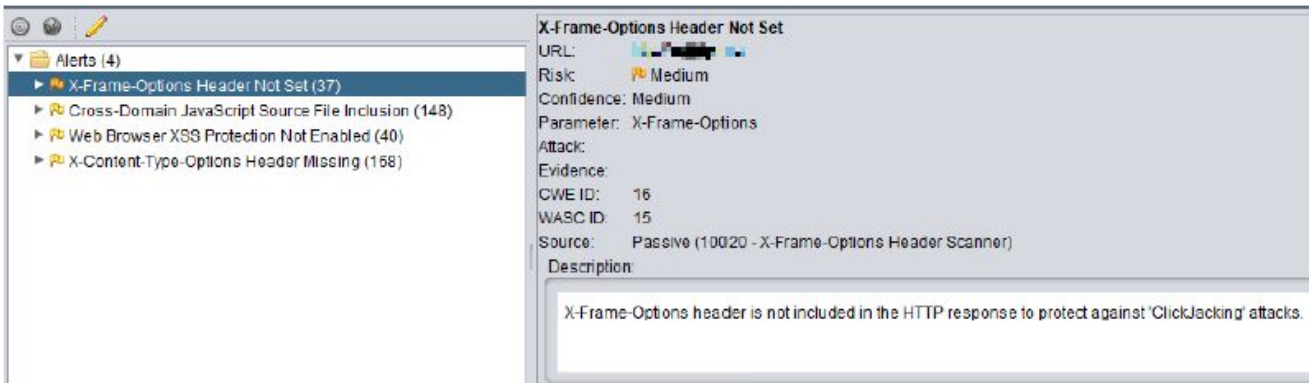


FIGURE 4.1 – Scan de vulnérabilités

Le détournement de clic, ou clickjacking, est une technique malveillante visant à pousser un internaute à fournir des informations confidentielles ou à prendre le contrôle de son ordinateur en le poussant à cliquer sur des pages apparemment sûres. Si l'entête « X-

frame-options » n'est pas défini, l'exploitation de cette vulnérabilité augmentera et peut avoir un impact sévère.

Remarque : Si notre méthodologie de développement sécurisée a été appliqué, les développeurs auraient évité cette faille. Il existe une recommandation citée dans notre "Guide du développeur", pour éviter ces genres de failles.

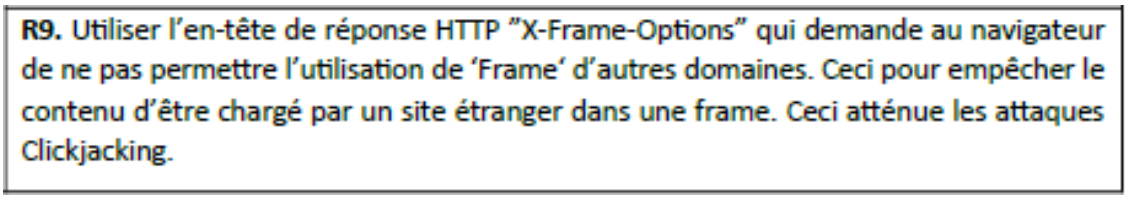


FIGURE 4.2 – Extrait du guide du developpeur

NB : Au niveau du site web, l'identification des vulnérabilités a été un vrai challenge, parcequ'il a été déjà testé à plusieurs reprises par l'équipe SOC et les vulnérabilités trouvées sont déjà remontées et résolues. Suite aux manques de vulnérabilités, nous ne pouvons pas mesurer de manière précise l'effectivité de notre méthodologie.

Heureusement un autre projet nous a été affecté pour pouvoir réaliser les tests. Ce projet n'a jamais encore été testé. C'est un code source écrit en Java. Le framework utilisé est Spring.

4.2 Analyse statique d'un code source

Durant cet analyse de type **white Box**, on mettera le point, seulement, sur les failles de risque majeur.

Voici un exemple des vulnérabilités de niveau majeur qui ont été trouvées à l'aide de l'outil FindsecBugs :

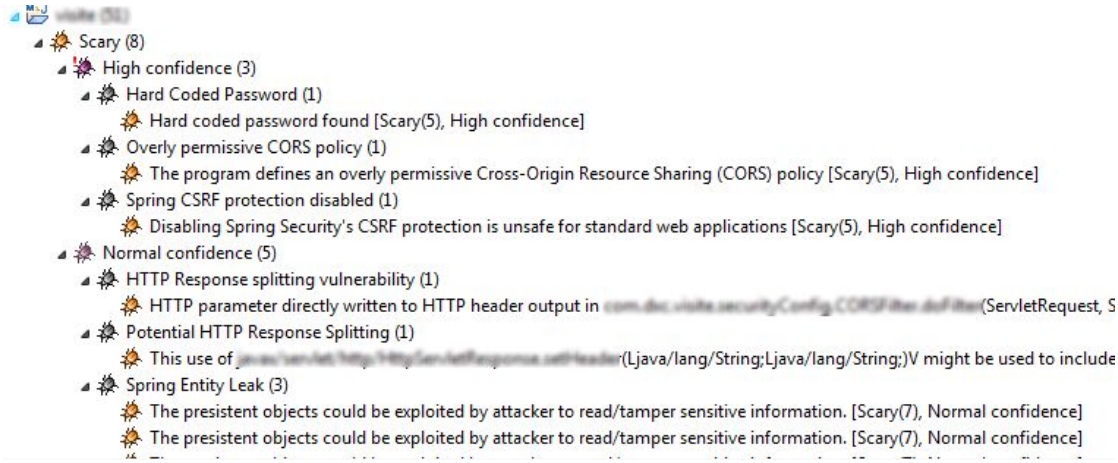


FIGURE 4.3 – Extrait de l'analyse statique

Exemple :

Les mots de passe enregistrés dans le code sont des cibles faciles pour repérer les mots de passe, permettant aux cybercriminels et aux logiciels malveillants de détourner des logiciels, des microprogrammes, des systèmes et des périphériques.

Remarque : Si notre méthodologie de développement sécurisée a été appliqué, les développeurs auraient évité cette faille. Il existe une recommandation citée dans notre "Guide du développeur", pour éviter ces genres de failles.

R1. Ne pas coder en dur les identifiants c.-à-d. ne jamais laisser les informations d'identification stockées directement dans le code de l'application.

FIGURE 4.4 – Extrait du guide du developpeur

4.3 Résultat de la simulation

Dans cette partie, nous allons prendre en considération que les résultats de l'analyse statique.

Voila un récapitulatif des failles de sécurité que nous avons remontés en faisant la revue de code à base d'outils Opensource d'analyse statique.

		Sécurité		
Resource	Description	Type	Niveau	Recommandation
.java	Add a nested comment explaining why this method is empty, throw	Code smell	Critical	Methods should not be empty
.java	Add a nested comment explaining why this method is empty, throw	Code smell	Critical	Methods should not be empty
.java	Add a nested comment explaining why this method is empty, throw	Code smell	Critical	Methods should not be empty
.java	Add a nested comment explaining why this method is empty, throw	Code smell	Critical	Methods should not be empty
.java	Add a nested comment explaining why this method is empty, throw	Code smell	Critical	Methods should not be empty
.java	Move constants to a class or enum.	Code smell	Critical	Constants should not be defined in i
.java	Replace this persistent entity with a simple POJO or DTO object.	Vulnerability	Critical	Persistent entities should not be use
.java	Replace this persistent entity with a simple POJO or DTO object.	Vulnerability	Critical	Persistent entities should not be use
.java	Hard coded parameter number		Critical	Passwords should not be kept in the
.java	The program defines an overly permissive Cross-Origin Resource Sharing (CORS) polic		Critical	Avoid using * as the value of the Acce
.java	Disabling Spring Security's CSRF protection is unsafe for standard web applications		Critical	A valid use case for disabling this pro
.java	A "NullPointerException" could be thrown; "obj" is nullable here.	Bug	majeur	Null pointers should not be derefere
.java	Call "Optional#isPresent()" before accessing the value.	Bug	majeur	Optional value should only be access
.java	Call "Optional#isPresent()" before accessing the value.	Bug	majeur	Optional value should only be access
.java	Call "Optional#isPresent()" before accessing the value.	Bug	majeur	Optional value should only be access
.java	Call "Optional#isPresent()" before accessing the value.	Bug	majeur	Optional value should only be access
.java	Define and throw a dedicated exception instead of using a generic c	Code smell	majeur	Generic exceptions should never be
.java	Define and throw a dedicated exception instead of using a generic c	Code smell	majeur	Generic exceptions should never be
.java	Remove this "Double" constructor	Code smell	majeur	Constructors should not be used to i
java	Remove this unused " " private field.	Code smell	majeur	Unused "private" fields should be re
.java	Remove this unused " " private field.	Code smell	majeur	Unused "private" fields should be re
.java	Remove this unused " " private field.	Code smell	majeur	Unused "private" fields should be re
java	Remove this unused " " private field.	Code smell	majeur	Unused "private" fields should be re
.java	Remove this unused " " private field.	Code smell	majeur	Unused "private" fields should be re
.java	Remove this unused private " " method.	Code smell	majeur	Unused "private" fields should be re
.java	This block of commented-out lines of code should be removed.	Code smell	majeur	Sections of code should not be comm
.java	This block of commented-out lines of code should be removed.	Code smell	majeur	Sections of code should not be comm
.java	This block of commented-out lines of code should be removed.	Code smell	majeur	Sections of code should not be comm

FIGURE 4.5 – Extrait du rapport de revue de code

En se basant sur la simulation réaliser par IBM sur le coût relatatif à la réparation des failles detectées en fonction des phases du cycle de développement de logiciel **[Figure 1.3]**, nous avons pu réaliser ce bilan :

Phase	Nombre de failles trouvé	Prix de réparation d'une faille (Estimation d'IBM)	Prix de réparation de toutes les failles
Conception		14 $	
Développement	28-1	30 $	810 $
Test		60 $	
Maintenance		110 $	
Total	27		810 $

FIGURE 4.6 – Bilan récapitulatif (La méthodologie est respectée)

Remarque : Dans un premier temps, nous avons choisi de faire une simulation dans la phase de développement. Déjà dans le cycle de développement de l'organisme d'acceuil, les tests de sécurité sont remportés jusqu'au phase des Tests. Donc chaque faille trouvé dans cette phase, nous faira économisé 30 $ et démontrera par la même occasion l'effectivité de notre méthodologie.

En ce qui concerne la phase de conception, généralement la revue d'architecture se fait en se basant sur des outils de Threat modeling. Le processus d'intégration de la sécurité dans la phase de conception est similaire.

Sans la méthodologie de développement sécurisé, le bilan sera comme suit :

Phase	Nombre de failles trouvé	Prix de réparation d'une faille (Estimation d'IBM)	Prix de réparation de toutes les failles
Conception		14 $	
Développement		30 $	
Test	28	60 $	1 680 $
Maintenance		110 $	
Total	28		1 680 $

FIGURE 4.7 – Bilan récapitulatif (La méthodologie n'est pas respectée)

Conclusion

Notre méthodologie se compose de deux parties : La sensibilisation et l'intégration des pratiques de sécurité au niveau de chaque phase de notre cycle SDLC. Nous avons remarqué que le guide du développeur nous a fait économiser 30 $ cette fois-ci. L'intégration des outils d'anlayse statique nous ferons économisé 870 $ (En se basant sur l'éstimation d'IBM).

Conclusion génerale

Ce travail fait la synthèse des différentes étapes que nous avons suivies pour integrer la sécurité en amont dans le cycle de développement de logiciel.

Ce projet consiste à améliorer la sécurité des logiciels produits, tout en donnant une réponse à la plus grande problématique du domaine qui est l'amélioration de la sécurité à moindres coûts : Nous avons mis en place une méthodologie de développement sécurisé que toutes les équipes du cycle de développement doivent suivre. La sensibilisation des parties prenantes, et l'intégration de la sécurité dans chaque phase (Définition de besoins, conception ...) de notre SDLC forment les deux piliers de notre méthodologie.

Les difficultés techniques majeures rencontrées durant le déroulement du stage étaient principalement liées à la bonne compréhension et assimilation d'un très grand nombre d'erreurs que les développeurs et les concepteurs fassent. Néanmoins, grâce à l'assistance et l'encadrement solide que nous avons reçus, on a pu affronter tous les obstacles rencontrés.

Les résultat des tests dans la partie simulation, nous ont montrés que nos guides et recommandations sont très utiles. L'integration de la sécurité dans la phase de développement a eu un très grand impact sur le coût gloale de réparation des failles.

Bibliographie

[W1] “Fiche de la société DXC Technology Maroc”, disponible sur : https://fr.wikipedia.org/wiki/DXC_Technology_Maroc

[W2] Site officiel de DXC Technology Maroc, disponible sur : http://dxc-technlogy-maroc.com

[W3] Statistique réalisée par "IBM System Institute", disponible sur : https://www.ibm.com/devops/method/experience/deliver/dibbe_edwards_devops_shift_left/

[W4] “Writing Secure Code”, M. Howard and D. LeBlanc, 2002, disponible sur : https://ptgmedia.pearsoncmg.com/images/9780735617223/samplepages/9780735617223.pdf

[W5] Iso 27034, disponible sur : https://www.iso.org/fr/standard/44378.html

[W6] Statique disponible sur : https://www.checkmarx.com/products/static-application-security-testing/

[W7] Statique disponible sur : https://www.veracode.com/products/binary-static-analysis-sast

[W8] Statique disponible sur : https://www.edgescan.com/wp-content/uploads/2018/05/2015-edgescan-Stats-Report-2015-v5.pdf

[W9] OWASP Top 10 des failles de sécurité, 20 juin 2017, disponible sur : https://www.owasp.org/images/f/f8/OWASP_Top_10_2017.pdf

[W10] “A methodology for secure software design”, E. B. Fernandez, 2004, documentation : https://pdfs.semanticscholar.org/f85e/5057b3ed20bea6a43ac12e470eb87e3117ff.pdf

[W11] ''A Survey on Design Methods for Secure Software Development",Amjad Hudaib, Mohammad Aref Alshraideh,Mohammed Alkhanafseh, December 2017, documantation sur : https://www.researchgate.net/publication/321886359_A_Survey_on_Design_Methods_for_Secure_Software_Development

[W12] Iso 3100, disponible sur : https://www.iso.org/fr/iso-31000-risk-management.html