REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEINGENEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE SAAD DAHLEB - BLIDA 1 -

FACULTE DES SCIENCES

DEPARTMENT D'INFORMATIQUE

Mémoire de fin d'études

Présenté en vue de l'obtention du diplôme de master

Domaine : Informatique

Filière : Informatique

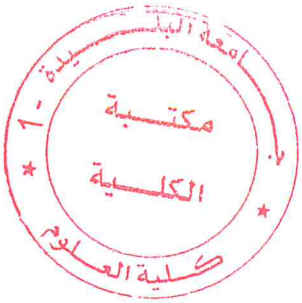
Spécialité : Sécurité des systèmes d'information

Thème : PROPOSITION D'UN SYSTÈME D'AUDIT CODE SOURCE

SESSION : JUIN 2017

Présenté par :

- **BENDIMERAD Adel Reda**
- **BENKOULAL Abdelghani**



Devant le jury :

Présidente de jury : Mme AROUSSI Sana

Examinateur : Mlle TOUBALINE Nesrine

Promotrice : Mme REZOUG Nachida

Encadreur : Mlle ADJAILIA Roumaissa

Dédicace

Je dédie ce modeste travail à titre posthume à mon défunt père que je n'ai pas eu la chance de connaitre et qui je l'espère sera fière de moi de là où il se trouve.

À ma mère qui représente ce que j'ai de plus chère en ce monde, qui a sacrifié sa vie pour faire de moi la personne que je suis, sa tendresse sa bien vaillance, son amour et son inconditionnelle présence à mes côtés m'ont permis de réussir et de gravir les échelons.

À mes deux grandes sœurs Soraya et Batoul qui m'ont soutenu et encouragé

À mes deux grand freres Zoheir et Mohamed el hadi qui ont su être là au bon moment

À ma chère sœur Samia, qui sans elle rien de concret n'aurait pu être fait, son encouragement et son soutien moral a été pour moi capital.

À mes deux meilleurs amis Chakib et Lilia qui ont toujours été là pour moi dans les bons comme dans les mauvais moments.

A tous mes ami(e)s et mes collègues.

Et enfin à toutes les personnes qui m'ont soutenue durant cette année.

BENDIMERAD ADEL REDA

Dédicace

Tous les mots ne sauraient exprimer la gratitude, l'amour, le respect, la reconnaissance, c'est tout simplement que : je dédie ce mémoire à :

À Ma tendre mère Amina : tu représentes pour moi la source de tendresse et l'exemple de dévouement qui n'a pas cessé de m'encourager. Tu as fait plus qu'une mère puisse faire pour que ses enfants suivent le bon chemin dans leur vie et leurs études.

À Mon très cher Père Ben Youcef : Aucune dédicace ne saurait exprimer l'amour, l'estime, le dévouement et le respect que j'ai toujours pour toi. Rien au monde ne vaut les efforts fournis jour et nuit pour mon éducation et mon bien-être. Ce travail et le fruit de tes sacrifices que tu as consentis pour mon éducation et ma formation le long de ces années.

À mes très chères grand-mères Malika et Houria et à toute ma famille.

À ma très chère fiancée Asmaa Haffis: ton soutien moral et ta foi en moi m'ont permis de réussir mes études. Je te dédicace ce travail en guise de remerciement et en témoignage de ma reconnaissance et de ma gratitude envers toi.

À mes futur beaux-parents.

À mes chers frères : Fethi, Wassim et Amine.

À mes très chers amis et à tous les membres de ma promotion et mes enseignants.

À tous ceux qui me sont chers et que j'ai omis de citer.

BENKOULAL ABDELGHANI

Sommaire

Liste des figures :

Listes des tableaux :

Résumé

La sécurité applicative demeure une priorité pour n'importe quelle entreprise et réaliser une sécurité complète et rigoureuse d'une application n'est pas toujours chose facile, presque chaque jour, des centaines de sociétés sont attaquées et les données personnelles sont compromises. L'objectif de ce travail est de proposer un système d'audit code source en utilisant l'analyse statique, cette technique permet d'analyser un programme sans toutefois l'exécuter. De nos jours, l'utilisation de l'analyse dynamique s'accapare tout le marché de la sécurité de l'information. Alors que l'analyse statique demeure encore peu connue et assez redoutée par les auditeurs et les spécialistes de la sécurité, certains spécialistes de l'audit proposent de combiner les deux méthodes afin d'avoir une complémentarité dans le résultats, l'analyse statique offre plusieurs aspects en matière de sécurité informatique, mais celui dont il sera surtout question dans ce rapport est la détection de vulnérabilités contenues dans le code source, ainsi que les différentes méthodes qui ont contribué à la réalisation, l'implémentation et à la mise en place de notre système d'audit code source.

Mots-clés : sécurité, audit, analyse statique, cycle de développement logiciel, code source, analyse dynamique.

Abstract

Application security remains a priority for any enterprise and achieving complete and rigorous security of an application is not always easy, almost every day hundreds of companies are attacked and personal data is compromised. The aim of this work is to propose a system of auditing source code using static analysis, this technique makes it possible to analyze a program without however executing it. Nowadays, the use of dynamic analysis takes over the entire information security market. While static analysis is still little known and dreaded by auditors and security specialists, some audit specialists propose to combine the two methods in order to have complementarity in the results, static analysis offers Several aspects of computer security, but the main focus of this report is the detection of vulnerabilities contained in the source code, as well as the different methods that have contributed to the implementation, implementation and implementation of our source code auditing system.

Keywords : security, audit, static analysis, software development cycle, source code, dynamic analysis.

Introduction générale :

L'informatique est devenue un outil incontournable de notre quotidien, elle est souvent utilisée pour gérer des organisations ou des millions de dollars et de vies humaines sont en jeu, en vue de cette grande importance, il est primordial de sécuriser ces données contre tous types de risques qui peuvent être d'origine naturelle ou humaine, accidentelle ou bien intentionnelle, car un seul problème peut conduire à une catastrophe d'une ampleur phénoménale.

La plupart des méthodes de détections et de réductions de vulnérabilités se déroulent lors de l'achèvement du processus de développement logiciel, qui veut dire que le produit logiciel devra être exécuté et confronté à une batterie de test, cependant, il existe des méthodes qui peuvent être réalisés à un niveau bien avancé du cycle de vie du développement logiciel, pendant la phase d'implémentation de ce dernier, il est recommandé d'effectuer une analyse statique du code source afin de détecter de potentielles vulnérabilités sans avoir à exécuter le code, cela va permettre d'augmenter le niveau de sécurité et la fiabilité du produit mais aussi de réduire drastiquement le cout et le temps de développement de ce dernier.

Dans ce document nous allons traiter les différents aspects de la sécurité applicative, mais nous allons nous concentrer beaucoup plus sur l'analyse statique qui consiste en gros à scanner et analyser le code source avant son exécution, et cela vu la grande importance de cette dernière dans le domaine de la sécurité.

Paradoxalement beaucoup d'informaticiens redoutent cette approche et cela est dû au fait que cette dernière est connu beaucoup plus comme étant une technique manuelle (l'analyse manuelle ou le code review en anglais), ces dernières années nous avons constaté l'apparition d'une nouvelle technique (analyse du code source) qui est la version automatique de l'analyse manuelle.

Même s'il existe des outils d'audit du code source sur le marché, l'analyse statique n'est pas aussi célèbre que l'analyse dynamique, cependant son point fort est la détection des vulnérabilités et des erreurs de programmation, ce qui permettra par la suite de réduire le coût et le temps de développement et de déploiement.

Le marché de l'analyse statique et de l'audit propose différents outils offrant ainsi une meilleure maîtrise de la sécurité de l'information et des données. Cependant, aucun outil ne permet d'être optimal par rapport aux résultats attendus, le système qu'on propose permettra de répondre aux exigences et de palier à certains manques notamment lors du traitement du résultat, la chose qui

le rend robuste est justement le fait qu'il soit modulable et extensible et qui permettra d'offrir une analyse efficace et complète lors de l'audit.

Dans le présent mémoire, nous proposons d'étudier chronologiquement les éléments suivants :

- Chapitre I : Intitulé « Introduction à la sécurité » Il se composera d'une partie généralité à propos de la sécurité de l'information, ainsi qu'une partie prospection marché.
- Chapitre II : « Les méthodes de tests » il se composera de deux parties où nous exposerons tout ce qui se rapporte à l'analyse statique du code source dans la littérature ainsi que les différents outils existants avec leurs points positifs et leurs inconvénients et les différentes techniques d'implémentation de la méthode d'analyse statique.
- Chapitre III : « Solution proposée » qui permettra d'exposer les besoins fixés par l'entreprise et de présenter notre solution et sa conception ainsi que l'architecture en module utilisée pour faciliter la réalisation.
- Chapitre IV : « Test et Validation » sera dédié à présenter l'environnement du travail tout en précisant les outils que nous utiliserons ainsi que l'exposition du test de notre système et sa validation par une équipe d'experts.

Chapitre I : Introduction à la sécurité

1 Introduction :

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 les 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.

Par ailleurs, les nouvelles tendances permettent, non seulement, aux utilisateurs d'avoir accès aux ressources, mais aussi de transporter une partie du système d'information en dehors de l'infrastructure sécurisée de l'entreprise. D'où la nécessité de mettre en place des démarches et des mesures pour évaluer les risques et définir les objectifs de sécurité à atteindre.

2 Les menaces informatiques :

La menace informatique représente le type d'actions susceptibles de nuire dans l'absolu à un système informatique. En termes de sécurité informatique, les menaces peuvent être le résultat de diverses actions en provenance de plusieurs origines (1) :

- Origine opérationnelle :

Ces menaces sont liées à un état du système à un moment donné. Elles peuvent être le résultat d'un bug logiciel (Buffer overflows, format string …etc.), d'une erreur de filtrage des entrées utilisateur (typiquement les XSS et SQL injection), d'un dysfonctionnement de la logique de traitement ou d'une erreur de configuration

- Origine physique :

Elles peuvent être d'origine accidentelle, naturelle ou criminelle. On peut citer notamment les désastres naturels, les pannes ou casses matérielles, le feu ou les coupures électriques.

- Origine humaine :

Ces menaces sont associées directement aux erreurs humaines, que ce soit au niveau de la conception d'un système d'information ou au niveau de la manière dont on l'utilise. Ainsi elles peuvent être le résultat d'une erreur de conception ou de configuration comme d'un manque de sensibilisation des utilisateurs face au risque lié à l'usage d'un système informatique.

3 Enjeux de la sécurité des systèmes d'information :

Le système d'information constitue un patrimoine essentiel des entreprises. Constitué d'un ensemble de ressources matérielle et logicielle, il permet de traiter, stocker et transférer les données des entreprises. Ainsi la sécurité des systèmes d'information cherche à apporter une meilleure maîtrise des risques qui pèsent réellement sur l'entreprise et répondre à certains enjeux qu'on peut résumer en 4 lettres « DICA » (disponibilité, intégrité, confidentialité et authenticité). (1)

- Disponibilité : garantir l'accès aux ressources, au moment voulu, aux personnes habilitées d'accéder à ces ressources.
- Intégrité : garantir que les données échangées sont exactes et complètes.
- Confidentialité : garantir que seules les personnes autorisées peuvent avoir accès aux données et aux ressources de l'entreprise.
- Authenticité : garantir l'authenticité de l'information en s'assurant que l'information ne soit pas modifiée par une tierce personne.

4 L'importance de la sécurité du logiciel :

Le logiciel est une partie intrinsèque de l'activité, à la fin du XXe siècle, pratiquement toutes les entreprises en dépendent, dans tous secteurs confondus : aide au développement, à la production, à la commercialisation et au soutien du produit et services.

Le rapport du Comité consultatif sur les technologies de l'information (PITAC) du président des États-Unis de 2005 a déclaré que « Les pratiques d'ingénierie de logiciel couramment utilisées laissent passer des erreurs dangereuses, telles qu'une mauvaise manipulation des débordements de tampon, ce qui permet à des centaines de programmes d'attaquer et de compromettre des millions d'ordinateurs chaque année. Cela se produit principalement parce que l'ingénierie logicielle commerciale manque aujourd'hui de contrôles rigoureux nécessaires pour développer des produits sécurisés de haute qualité avec un coût acceptable. » (2)

Depuis la fin des années 1990, où la discipline de la sécurité du logiciel est apparue, jusqu'à aujourd'hui, où elle représente mondialement le sous-domaine de la sécurité des technologies informatique ayant la plus forte croissance, la sécurité applicative a grandement évolué…

5 Qu'est-ce que la sécurité applicative ?

La sécurité applicative est un processus effectué pour appliquer des contrôles et des mesures aux applications d'une organisation afin de protéger les données contre l'accès, l'utilisation, la diffusion, la destruction, ou la modification non autorisée, 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. (3)

La sécurité applicative est au centre des nouvelles menaces et des intrusions récentes : aujourd'hui, les attaques ciblant directement les applications sont de plus en plus nombreuses. Malheureusement, les équipes de sécurité sont historiquement centrées sur la protection des infrastructures et souvent très éloignées des problématiques applicatives.

5.1 Des statistiques sur la sécurité applicative :

Nous allons présenter dans ce qui suit quelques statiques à propos de la sécurité applicative (4) :

- **84%** des attaques ciblent la couche applicative « Checkmarx 2015 »
- **75%** des vulnérabilités se retrouvent dans la couche applicative « Checkmarx 2015 »
- **70%** des applications avaient au moins une vulnérabilité classifiée dans le top 10 *OWASP* « Veracode 2015 »
- **15%** des applications web ont un taux de vulnérabilité critique ou élevé « Edgescan report 2015 »
- Dans certains domaines de l'industrie, les applications web sont responsables de **35 %** des brèches « Verizon DBIR 2015 »
- **1 transaction sur 86** des détaillants web est une tentative de fraude, une augmentation de **30 %** par rapport à 2014 « ACI Worldwide 2015 »
- **250 $** : coût moyen par enregistrement perdu ou volé « 2015 Cost of Data Breach Study: Canada », Ponemon Institute –

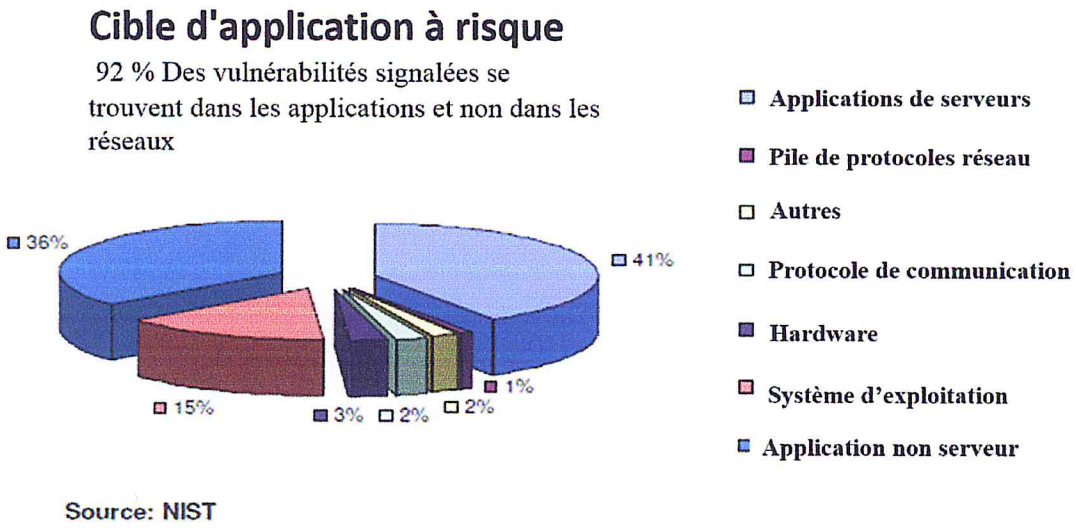


Figure 1: Le taux de vulnérabilités détectées dans les différents domaines informatiques. (5)

Selon le National Institute of Standards and Technology, ou NIST (qu'on pourrait traduire par « Institut national des normes et de la technologie »), est une agence du département du Commerce des États-Unis. Son but est de promouvoir l'économie en développant des technologies (6), a affirmé que les vulnérabilités détectées concernent à 92% les applications et non pas les réseaux (address hips, firewalls,ips,ids…) comme la majorité semble le croire.

Une étude internationale a interrogé 450 décideurs informatiques et révèle que de nombreuses sociétés se heurtent aux exigences de gouvernance et de sécurité des échanges de données, mais pas que, l'étude a aussi révélée les quelques points ci-dessous (5):

- 23% des entreprises ont récemment échoué à un audit de sécurité, tandis que 17 % doutent de leur capacité à réussir un audit de conformité des échanges de données.
- Le coût total moyen d'une atteinte à l'intégrité des données s'élève à 2,4 millions d'euros.
- La stratégie d'intégration n'est pas alignée avec les structures et les politiques de gouvernance, de confidentialité et de sécurité des données pour 71% des entreprises.

Les attaquants peuvent potentiellement utiliser différents chemins à travers les applications pour porter atteinte aux métiers ou à l'entreprise, chacun de ces chemins représente un risque qui peut, ou pas, être suffisamment grave pour mériter une attention particulière.

L'erreur informatique est une déficience ou une erreur dans un programme qui l'empêche de fonctionner normalement, cependant ces bugs peuvent aller d'un simple arrêt d'un logiciel dans votre ordinateur à des conséquences beaucoup plus catastrophiques surtout que le bug informatique a causé d'énormes pertes humaines et matérielles durant le siècle dernier, par ailleurs, le problème majeur des programmes est leur instabilité : la modification d'une petite partie du code se propage en chaîne avec des effets très difficiles à déterminer ;

Les vérificateurs formels déterminent si les formules logiques gigantesques ainsi obtenues sont vraies ou fausses, ce qui est hors de portée pour un homme.

5.2 L'OWASP (Open Web Application Security Project):

L'OWASP pour Open web Application Security Project est une organisation à but non lucratif. Sa mission principale consiste à proposer aux internautes, administrateurs et entreprises des méthodes et outils de référence permettant de contrôler le niveau de sécurisation de ses applications web. L'OWASP publie régulièrement des recommandations de sécurisation elle est devenue une référence pour tous ceux qui sont intéressés par la sécurité applicative. (7)

5.3 Classification des vulnérabilités d'OWASP (top 10) :

Le Top Ten d'OWASP représente un large consensus sur les défauts de sécurité les plus critiques à propos des applications web (figure 2). Un large éventail d'entreprises et d'agences du monde entier utilisent l'OWASP Top Ten comme norme pour sécuriser les applications. Selon Veracode **70%** des applications avaient au moins une vulnérabilité classifiée dans le top 10 *owasp* ce qui représente un nombre très important. (8) Par exemple, sur le marché commercial, la norme PCI (Payment Card Industry) a adopté le Top Ten d'OWASP et exige (entre autres) que tous les commerçants obtiennent une révision du code de sécurité pour tous leurs codes personnalisés.

OWASP Top 10 – 2010 (Précédent)	OWASP Top 10 – 2013 (Nouveau)
A1 – Injection	A1 – Injection
A3 – Violation de Gestion d'authentification et de Session	A2 – Violation de Gestion d'authentification et de Session
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Références directes non sécurisées à un objet	A4 – Références directes non sécurisées à un objet
A6 – Mauvaise configuration sécurité	A5 – Mauvaise configuration sécurité
A7 –Stockage cryptographique non sécurisé – Fusionné avec A9 →	A6 – Exposition de données sensibles
A8 – Manque de restriction d'accès à une URL – Elargie dans →	A7 – Manque de contrôle d'accès au niveau fonctionnel
A5 – Falsification de requête intersites (CSRF)	A8 – Falsification de requête intersites (CSRF)
<inclus dans A6: Mauvaise configuration sécurité>	A9 – Utilisation de composants avec des vulnérabilités connues
A10 – Redirection et Renvois non validés	A10 – Redirection et Renvois non validés

Tableau 1: Le TOP 10 des vulnérabilités de sécurité des applications web d'Owasp. (9)

5.4 Objectif de la sécurité applicative :

L'objectif principal de la sécurité applicative est donc justement d'éviter les failles informatiques (bugs) en ayant une méthodologie de développement, contrôles et tests ainsi qu'un plan de migration des applications, afin de minimiser les pertes humaines et financières en premier lieu et atténuer le vol d'information en deuxième lieu, ce qui n'a pas été le cas durant les deux dernières décennies et qui a provoqué de lourdes pertes comme montrées dans le tableau ci-dessous.

	Airbus A320 (1993)	Ariane 5 Galileo Poseidon flight 965 (1996)	Lewis Pathfinder USAF STEP (1997)	Unit 2 Delta 3 Near (1998)	DS-1 Orion 13 Galileo titan 4B (1999)
Coût agrégé	/	**640 Million $**	**116,8 Million $**	**225 Million $**	**1,6 Milliard $**
Pertes humaines	**3**	**160**	/	/	/

Pertes de données	/	**OUI**	**OUI**	**OUI**	**OUI**

Tableau 2: Pertes aérospatiales récentes dues à des défaillances logicielles. (5)

Les initiatives de sécurisation se multiplient (méthodologie de développement, gestion des Identités, analyses de risques projets, web application firewall, tests…) mais montrent rapidement leurs limites du fait de cette distance et des incompréhensions qui en découlent.

Il convient alors d'assurer une cohésion des activités de sécurité applicative, de systématiser celles-ci au cœur même des projets, et d'industrialiser la mise en œuvre de ces activités pour réduire les coûts et améliorer le niveau de sécurité global.

La sécurité applicative ne couvre pas uniquement la portion logicielle, elle couvre également tous les contrôles et mesures impliqués dans le cycle de vie de l'application et cela nécessite d'*intégrer* la sécurité applicative le **plus tôt possible** dans le cycle de vie de développement d'un système(SDLC).

6 SDLC (System Development Life Cycle) :

La méthodologie SDLC (cycle de vie de développement du système) est une approche structurée pour le développement de systèmes informatisés, elle consiste à définir un plan détaillé qui décrit comment développer, maintenir, remplacer et modifier ou améliorer certains logiciels (10). En adoptant cette méthodologie, les développeurs peuvent produire des systèmes plus fiables d'une façon plus rapide et plus économique (11). Avec d'autres mots les SDLCs sont utilisés pour pouvoir produire plus de fonctionnalité, avec une grande qualité, dans un laps temps court, avec moins de ressources et d'une manière prédictible (12).

La figure suivante montre les étapes d'un SDLC typique en commençant par la planification et se termine par le déploiement :

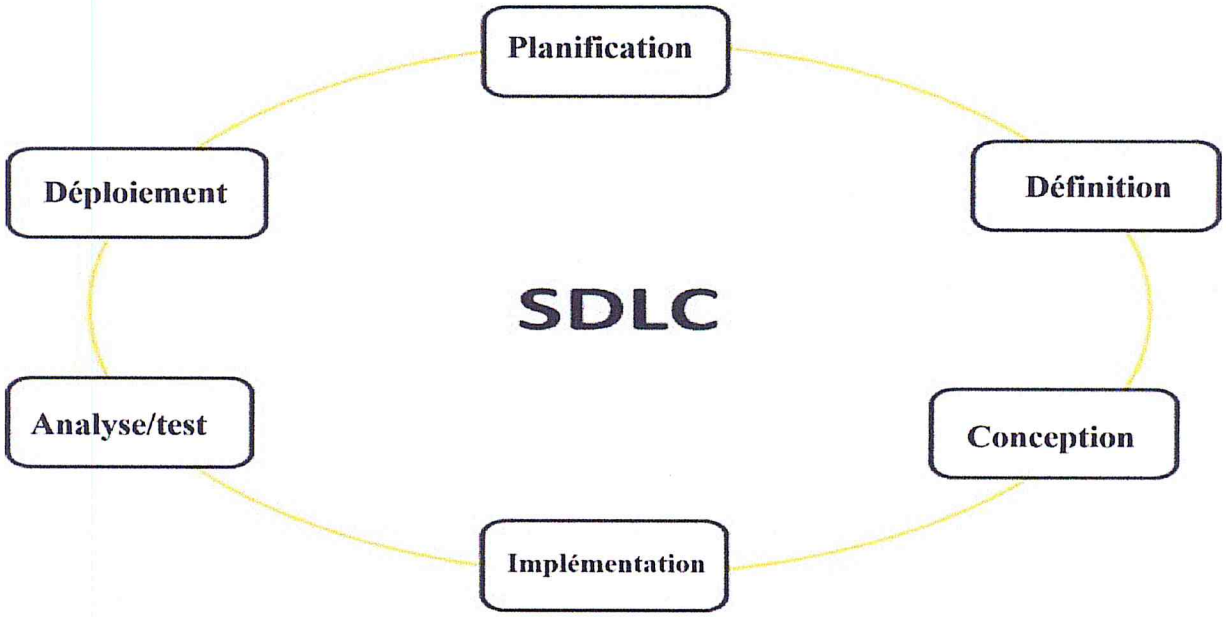


Figure 2: Cycle de vie de développement d'un système (SDLC). (10)

6.1 Les cycles de développement sécurisé (SDL) :

Généralement la phase de test est dans les dernières étapes du SDLC, dans cette partie nous allons voir comment introduire la sécurité dans le SDLC et pourquoi, et nous verrons aussi le modèle de cycle de développement sécurisé de Microsoft.

6.1.1 **Définition :**

Le SDL (Security development life cycle) la sécurité du cycle de vie de développement logiciel est une extension de la méthodologie SDLC, dans les modèles SDL la sécurité est introduite dans chaque phase du SDLC, la Figure 4 ci-dessous nous montre quelles sont les mesures de sécurité à prendre en compte dans chaque phase du SDL. (13)

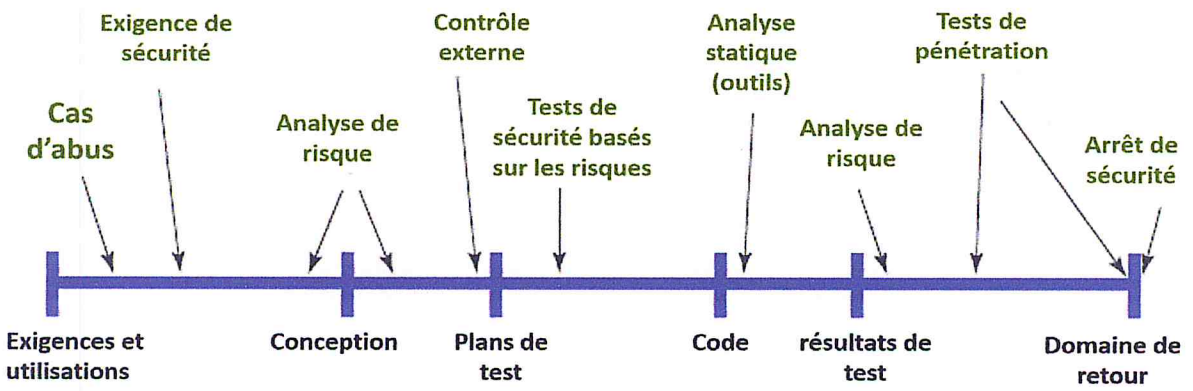


Figure 3 : sécurité du cycle de vie de développement (SDL). (13)

Comme nous l'avons vu précédemment le taux élevé de vulnérabilités a besoin d'être réduit, c'est pour cela que les SDL ont été créés. Il existe plusieurs modèles pour la méthode SDL, l'un des plus connus est le MICROSOFT SDL, mais il existe beaucoup d'autres comme l'Adobe Secure Product Lifecycle (Adobe, 2011) et le Cisco Secure Development Lifecycle (CISCO, 2011).

6.1.2 **Raisons principales pour la réalisation d'un SDL :**

L'utilisation d'un SDL lors du développement d'un produit logiciel est très importante, car ce dernier permet de réduire le coût et d'augmenter la sécurité du produit. Nous allons voir comment l'utilisation d'un SDL permet cela :

a) Réduction du coût :

La NIST (National Institute of Standards and Technology) a publié dans un rapport appelé 'The Economic Impacts of Inadequate Infrastructure for Software Testing'[10], le tableau 2 suivant :

Phase d'introduction	Spécification des besoins	Codage / test unitaire	Intégration	Test	Après déploiement
Spécification des besoins	2h	4h	6h	8h	10h
Codage / test unitaire	Non	2h	4h	6h	10h
Intégration	Non	Non	4h	8h	16h

Tableau 3: Le nombre d'heures nécessaire pour la correction d'un bug. (5)

Le tableau 3 montrée ci-dessus présente le nombre d'heures nécessaires pour fixer un bug selon la phase où il est introduit. On constate que plus la découverte du bug se fait dans une phase tardive que la phase où il a été introduit plus le nombre d'heures est grand ce qui implique un coût plus élevé. Par exemple si un bug est introduit dans la phase d'intégration, le bug prendra 2 heures pour être corrigé dans la même phase et on voie que le nombre d'heures augmente après chaque phase.

[1]

[1] **L'attribut 'Non' : Non applicable, impossible de détecter un bug avant son introduction.**

Le SDL introduit la sécurité dans chaque phase du cycle du début à la fin, ce qui permet la détection de beaucoup de bugs dans les phases où ils sont introduits et ce qui implique un coût de développement plus faible.

b) Augmentation de la sécurité logicielle :

Comme nous l'avons déjà vu le SDL introduit la sécurité dans toutes les phases du cycle de développement cela implique évidemment un produit logiciel plus amplement sécurisé.

6.2 Le cycle de développement sécurisé de Microsoft :

Le cycle de développement sécurisé (SDL), est un processus de développement logiciel proposé par Microsoft pour aider les développeurs à réaliser des logiciels plus sécurisés et qui répondent aux exigences de conformité de sécurité, tout en réduisant le coût de développement du logiciel, les SDL introduisent la sécurité dans toutes les phases du processus, comme montré dans la figure 5

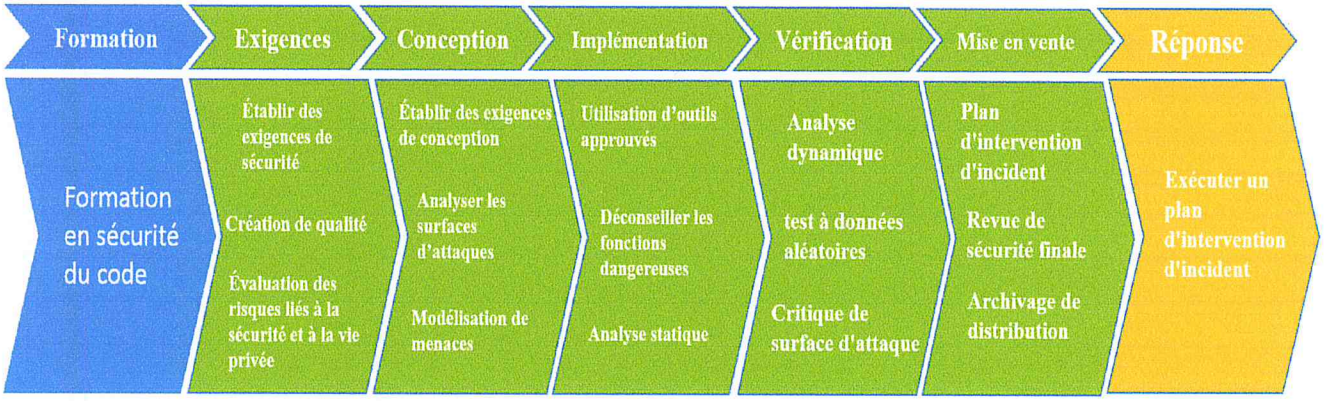


Figure 4: Microsoft Security Development Life cycle (SDL). (14)

7 Le Marché des outils de l'analyse statique du code source :

Dans le dernier rapport du *Magic Quadrant 2015*, Gartner a identifié les façons dont la sécurité est testée dans les applications et les principales sociétés proposant des solutions de test. En plaçant les entreprises dans son rapport, *Gartner* s'est concentré sur la maturité d'un fournisseur en proposant des fonctionnalités SAST et DAST en tant qu'outil ou en tant que service de sécurité et s'est basé sur certain nombre de critères comme la commercialisation, le service à la clientèle, l'efficacité opérationnelle, la compréhension du marché, l'innovation mais aussi les stratégies de marché et de vente. (15)

Comme montré dans la figure 6, le rapport Gartner a classifié en deux principales catégories les différentes entreprises qui développent et vendent les outils de test de sécurité des applications

en les comparants sur le plan de deux critères qui sont la complétude de vision des entreprises dans les applications qu'ils proposent, ainsi que la capacité d'exécution.

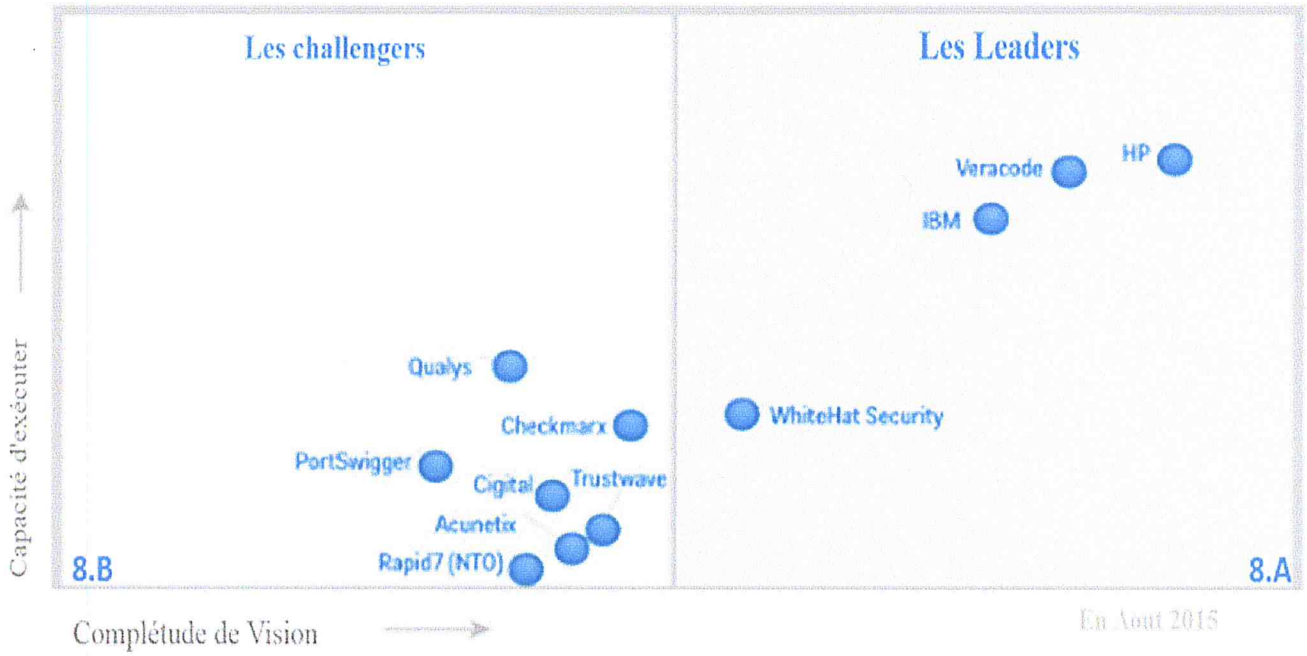


Figure 5 : classification du Garner Magic Quadrant 2015 test de sécurité des applications (AST). (15)

Chaque année les entreprises changent de catégorie en fonction des améliorations et des nouveautés qu'ils apportent à leurs outils de test, que ce soit en apportant une meilleure complétude de vision ou bien une capacité d'exécution plus performante.

7.1 Les catégories :

Les quatre catégories qui ont été choisies par Gartner afin de classer les fournisseurs sont :

7.1.1 Les Leaders (8.A) :

Les entreprises dans le quadrant *Leader* du MQ ont la plus haute capacité à exécuter et ont une vision complète. Gartner a identifié quatre leaders de test d'application dans le MQ qui offrent différentes fonctionnalités comme illustré dans le tableau 4 ci-dessous.

	Minimisation de faux positives	Technologie IAST	Analyse statique du code	Test de sécurité d'application mobile	Test manuel

HP	Oui	Oui	Non/ Oui	Oui/limité	Non/ Oui
White Hat Security	Oui	Non	Oui	Non	Oui
Veracode	Oui	Oui/service	Oui/Service	Oui/Service	Oui
IBM	Oui	Oui	Oui	Oui (Service cloud)	Oui

Tableau 4: comparaison des différentes fonctionnalités proposées par les leaders de AST.

Le SAST de HP dispose d'un support linguistique très étendu mais cet outil reste encore accessible qu'en offre premium (version payante)., il y'a aussi Veracode qui propose presque toutes ses technologies comme des services jusqu'à pouvoir offrir des outils complets dans un futur proche. D'autre part IBM possède une solide capacité de marché, avec son Security AppScan, reconnu pour ses capacités de test de sécurité des applications d'entreprise, on a aussi WhiteHat Security qui se distingue par son accent sur l'élément humain dans son offre, les résultats de toutes les analyses DAST et SAST de WhiteHat sont examinés par un expert humain avant la livraison au client.

Nous remarquons que tous les vendeurs d'outils proposent la fonctionnalité de minimisation de faux positives, cependant le taux de détections est relativement faible et varie d'un outil a un autre, de plus le test manuel est primordiale, car il sert à réduire les faux positifs d'une façon plus efficace.

Ainsi comme montré dans la figure 7, les différents fournisseurs qui proposent la méthode d'analyse statique de code source et nous remarquons que les quatre entreprises sont aussi leaders en ce qui concerne cette méthode qui est selon Gartner, la plus prometteuse pour détecter les vulnérabilités au niveau du code source.

De plus, ces outils disposent de bien d'autres solutions et fonctionnalités :

- **Acunetix :** sa solution peut analyser des applications JavaScript complexes côté client pour tester les vulnérabilités. Gartner met en garde que l'entreprise ne dispose pas des capacités SAST. Aucune fonctionnalité spécifique de test d'application mobile n'est proposée par l'entreprise.
- **Checkmarx :** Il dispose de l'une des technologies SAST les plus performantes. Cependant Gartner note que Checkmarx n'offre pas son propre DAST pour les applications web, mais que son DAST pour .NET et Java est en phase de planification.
- **Cigital :** dispose d'un outil innovant SecureAssist qui permet de vérifier un nombre limité de problèmes de codage SAST en s'intégrant directement dans l'environnement de développement intégré (IDE) du développeur.
  Gartner avertit que SAST de Cigital en tant que service est une offre moins connue et n'a pas encore été largement adoptée.

➢ **Remarque :** les entreprises du quadrant *(Challengers)* ont une grande capacité d'exécution mais une vision moins complète par rapport aux leaders qui offrent plus de fonctionnalités que les challengers, cependant toutes les entreprises proposent une fonctionnalité de minimisation de faux positifs mais le taux de détection reste plus bas que celui des leaders.

8 Conclusion :

Nous avons vu dans cette partie de l'état de l'art quelques généralités sur la sécurité de l'information, nous avons vu aussi l'importance que représente la sécurité applicative dans le domaine informatique en général et dans la sécurité de l'information en particulier, nous avons également mis l'accent sur le cycle de développement sécurisé et à quel moment et dans quelle phase il rentre en jeu lors du développement logiciel, et pour finir nous avons réalisé une prospection marché des outils d'audit code source et de leurs concepteurs, cela a démontré que différentes solutions existaient mais qu'elles présentaient certaines incomplétudes et plusieurs insuffisances dans ce qu'ils proposent comme fonctionnalités.

Chapitre II : Les méthodes de tests

1 Introduction :

Dans cette deuxième partie de notre chapitre nous allons nous intéresser essentiellement aux différentes méthodes existantes qui ont permis l'implémentation des analyseurs statiques du code source, nous verrons aussi que mise à part l'analyse statique, d'autres méthodes de test de sécurité existent offrant ainsi un large panel de possibilités en matière de test, d'audit et de sécurisation des données.

2 Les méthodes de test :

Dans la partie précédente nous avons vu le SDL de Microsoft (Figure 5), dans cette partie nous nous intéresserons à deux pratiques particulières dans le SDL, la première est la réalisation d'une analyse statique dans la phase quatre, la deuxième est la réalisation d'une analyse dynamique dans la phase cinq, ces deux pratiques sont utilisées dans le SDL, ce sont deux pratiques fondamentales dans les tests informatiques. Dans cette partie nous allons voir ces deux méthodes tout en mettant l'accent sur l'analyse statique sachant que ce document traite principalement de l'analyse statique du code source.

3 L'analyse statique :

L'analyse statique ou l'analyse statique du code source (SAST) consiste à analyser le texte d'un programme pour en extraire de l'information. Cette analyse est effectuée sans exécuter le programme, ainsi l'analyse statique est utilisée pour repérer des erreurs de programmation, cette dernière est appliquée dans la phase d'implémentation et existe en deux versions, la version manuelle et la version automatique. (17)

En 1936 les travaux de recherche d'Alan Turing ont démontré qu'il n'existe aucun programme qui peut toujours répondre sans se tromper qu'un autre programme puisse ou non produire des erreurs à l'exécution, en d'autres termes le problème de l'arrêt est indécidable. (18)

Bien qu'il soit impossible de détecter toutes les erreurs présentes dans tous les programmes ; il est possible dans la majorité des cas de détecter une grande quantité des problèmes présents dans le code grâce à l'analyse statique, pour cela il faut utiliser des méthodes dites approximatives, qui veut dire qu'elles sont fiables sur la majorité des cas réels sans nécessairement être optimales.

D'après l'Open web Application Security Project (OWASP), l'analyse statique lorsqu'elle est utilisée avec des outils automatisés et des tests de pénétration manuelle, l'examen de code peut considérablement augmenter la rentabilité d'effort de vérification de la sécurité des applications, comme expliquée en quelques points ci-dessous :

a) Objectif :

Le but principal de cette méthode est de détecter les bugs et les failles de sécurité et de les corriger dans la phase de développement, d'après le tableau 2 vu dans la première partie du chapitre, si un bug est introduit dans la phase de spécification des besoins ou de l'implémentation, et qu'il est détecté dans cette dernière, cela ne prendra respectivement que 4H ou 2H pour le corriger, ce qui implique un coût plus faible et une meilleure qualité des produits informatiques.

b) L'audit du code manuel :

L'audit manuel ou la revue du code (code review), est une pratique qui consiste à lire le code source ligne par ligne par un auditeur à la recherche des bugs et des erreurs de programmation, mais cette dernière est devenue très couteuse en temps, effort et argent car les programmes sont devenus de plus en plus conséquents ce qui implique un très grand nombre de lignes de code (on parle de milliers de lignes de code). (19)

c) L'audit du code automatique :

L'audit du code consiste à scanner tout le programme à l'aide d'un outil automatisé, à la fin de l'audit l'outil retourne un rapport qui contient les portions de code infectées et les types de failles de sécurité et bugs trouvés, cela rend le travail de l'auditeur plus simple et pour cela il existe plusieurs outils qui sont en open source ou sous licence et qui ont leurs propres forces et faiblesses. (19)

c.1) faux positifs : On parle de "faux positif " lorsque un paquet anodin (ne consistuant aucune menace réelle) est détecté comme tentative d'intrusion. (20)

c.2) faux négatifs : On parle de faux négatif lorsqu'une tentative d'intrusion n'est pas détectée. (20)

d) L'audit du code manuel VS automatique :

L'audit du code source automatique est très puissant, mais il a toujours besoin du facteur humain pour confirmer les failles détectées, vu que ces outils peuvent générer de faux positifs et des faux négatifs. Par ailleurs, l'audit automatique ne prend pas en considération le contexte, par exemple si un programmeur a remplacé par erreur un « 2 » par un « 3 » dans une formule mathématique, l'outil ne peut pas détecter cette erreur et ne sera pas considéré comme telle, autre problème, si

un programmeur oublie une fonction par exemple la fonction d'authentification, l'outil ne pourra pas détecter l'erreur.

L'analyse manuelle a l'inconvénient d'être longue et couteuse, cependant elle n'a pas été abandonnée pour autant au profit de l'audit automatique, l'audit manuel est toujours utilisé car il peut détecter des erreurs que la version automatique ne peut pas détecter, en effet la méthode manuelle possède de nombreux avantages tels que la prise en considération du contexte et la logique du métier qui la rendent indispensable lors de l'analyse statique.

Pour avoir un résultat plus complet, l'utilisation de ces deux méthodes est essentielle du fait de leurs complémentarités.

e) Les outils d'analyse statique :

Le premier outil utilisé pour l'analyse statique est la commande « grep » de Linux, elle permet de faire une recherche de chaîne de caractères. (21) Dans le tableau 6 suivant, nous allons exposer quelques outils d'audit :

Nom	Langage supporté	Open Source / sous Licence
Rips	PHP	Open Source
Find Bugs	JAVA	Open Source
Visual Code Grepper	C/C++, C#, VB, PHP, JAVA, PL/SQL	Open Source
SonarQube	Multi langage	Open Source
Veracode	Multi langage	Sous Licence
HP Fortify	Multi langage	Sous Licence

Tableau 6: Liste des outils d'audit de code source open source et sous licence.

f) Le point faible principal de l'analyse statique :

Les outils d'analyse statique rencontre régulièrement des situations où ils ne peuvent pas déterminer précisément si une erreur est présente ou non, donc ils peuvent signaler des erreurs là où il n'y'en a pas et ne pas les signaler là où il y'en a. alors est que ces outils produisent trop de

faux positifs, également appelés fausses alarmes. Dans ce contexte, un faux positif est un incident détecté alors qu'aucun problème n'existe pas en réalité. Un grand nombre de faux positifs peut être un obstacle à l'utilisation d'outils d'analyse statique.

3.1 Les méthodes de l'analyse statique :

Les solutions trouvées dans la littérature afin de mettre en place un système de détection de vulnérabilités dans le code source ne sont pas nombreuses et cela est dû au fait que le problème qui doit être traité par l'analyse statique est indécidable comme nous l'avons vu précédemment, cependant plusieurs solutions ont été proposées afin de minimiser les erreurs et d'optimiser le résultat et la plupart de ces solutions sont des techniques utilisées entre autres dans le domaine de compilation, nous allons-en voir quelques-uns :

- **L'analyse lexicale :**
  Elle convertit la syntaxe du code source en des entités lexicales dans le but d'abstraire le code source et de le rendre plus facile à manipuler. (22)

- **Le string matching :**
  C'est une simple comparaison de chaînes qui peut être utilisée à des fins différentes, par exemple on peut faire correspondre les noms de fonctions dans le code source à une liste de fonctions indésirables. Cette technique peut être utilisée pour identifier les fonctions obsolètes et/ou non sécurisées. (23)

- **Le flux de données :**
  L'analyse de flux de données permet de recueillir des informations sur l'exécution des logicielles alors qu'il est dans un état statique. (24)

- **Le taint analyse :**
  Cette technique consiste à marquer les variables qui sont influencées par l'utilisateur par exemple dans un formulaire d'un site, on trouve un champ « Email » ou l'utilisateur va entrer sont Email, la valeur entrée sera manipulée par une variable dans le code source, cette dernière sera marquée (taguée) pour voir en suite si son utilisation dans le code source peut constituer un risque pour l'application ou non. (25)

- **Combinaison d'analyse statique et du data mining :**
  Le data mining est utilisé pour distinguer les faux positifs en utilisant les 3 principaux classificateurs d'apprentissage machine et pour justifier leur proximité en utilisant un classificateur d'acceptation. Le mélange des procédures de détection ne peut pas fournir des résultats entièrement corrects. Et le data mining ne permet pas de résoudre le problème mais

tout simplement donner des résultats probabilistes. L'outil corrige le code en insérant des correctifs, c'est-à-dire des fonctions de purification et de validation. Les tests sont utilisés pour vérifier si les correctifs évacuent vraiment les vulnérabilités et ne compromettent pas le comportement (correct) des applications. (26)

➢ **Autres algorithmes et structures de données qui sont utilisés dans le domaine de l'analyse du code statique :**

- **Arbre syntaxique abstrait :**

Lors de l'analyse du code de programme, la structure générée est un arbre abstrait. Il s'agit de la structure de données de base pour toutes les techniques d'analyse de code, ainsi que les techniques de compilation. Les langages sont habituellement décrits comme des grammaires sans contexte, ces grammaires peuvent être représentées comme un arbre, où chaque nœud correspond à une expression. (27)

- **Formulaire statique d'affectation unique (static single assignment SSA) :**

Le formulaire SSA est une propriété des représentations de programme qui indique qu'une variable ne peut être affectée qu'une seule fois n'importe où dans le programme. Cela signifie que plusieurs affectations à la même variable créent de nouvelles versions de cette variable et pratiquement de nouvelles variables. Cette représentation fait habituellement partie des représentations intermédiaires, et a une grande importance dans la simplification de certains algorithmes d'analyse, comme la propagation constante. (28)

- **Analyse de Pointeurs et d'Alias :**

Pointer Analysis (ou Points-to) vient répondre à la question à laquelle un pointeur pointe vers une zone mémoire. Analyse d'Alias répond à la question lorsque deux pointeurs pointent vers la même zone en mémoire. Comme mentionné dans les analyses antérieures, ceux-ci ont également des configurations et des niveaux de précision différents : sensibilité au flux, interprocessus, conscience du contexte (plus de facilités à gérer les fausses alarmes). Ces analyses sont les plus avancées jusqu'à nos jours, et aussi les plus compliquées. La plupart des algorithmes qui existent utilisent une sorte d'approximation.

La majorité des outils d'analyse statique utilise une combinaison de ces techniques et d'autres (qui ne sont pas citées dans ce rapport) pour trouver le maximum de vulnérabilités avec un taux de faux positif et de faux négatif assez faible. (29)

3.2 Les faiblesses détectées par l'analyse statique automatique :

Ce type de méthode permet la détection d'un grand nombre de bugs et de failles de sécurité, nous allons citer quelques-uns dans ce chapitre :

3.2.1 Les bugs :

Les outils d'analyse statique permettent la détection d'un nombre important de bugs qui peuvent conduire à l'arrêt de l'application lors de son déploiement. Ces outils peuvent détecter des bugs comme la division par zéro, Exception de pointeur null...

3.2.2 Les Injections :

Les attaques par injection consistent à introduire des portions de code malveillant dans les champs de saisies de l'application, par exemple dans un formulaire d'inscription d'un site web le pirate peut introduire un code malveillant dans l'email ou dans le mot de passe pour exécuter une action par le serveur pour son profil. Il existe plusieurs types d'injection en voici quelques-uns :

3.2.2.1 SQL injection :

Le pirate informatique introduit dans un champ de saisi une commande SQL pour la faire exécuter par le SGBD, si l'attaque réussie, le pirate informatique peut prendre possession de toute la base de données et accéder à toutes les informations de cette dernière. (30)

3.2.2.2 Cross site scripting (XSS) :

Les attaques de type XSS permettent d'injecter du contenu dans une page qui provoque par la suite des actions sur le navigateur. Pour réaliser une attaque de ce type, l'attaquant peut utiliser tous les langages qui sont pris en charge par le navigateur comme HTML, JAVASCRIPT, FLASH, … et bien d'autres. Avec cette technique l'attaquant peut rediriger l'utilisateur vers un site d'hameçonnage ou de voler les cookies, l'utilisation de SSL, ou d'autres mécanismes de chiffrement n'élimine ce type d'attaque. (31)

3.2.2.3 Injection de commande :

Quand un programme utilise une commande système et que cette dernière implique en partie une saisie utilisateur, une attaque est susceptible d'y injecter des instructions arbitraires. Tous les langages de programmation sont concernés, mais on observe souvent cela dans les scripts CGI écrits en Perl, PHP et shell ; moins en Java, Python ou C#. (32)

3.3 L'utilisation des fonctions dangereuses ou obsolètes :

Dans chaque langage il existe des fonctions qui sont très sensibles et qui peuvent conduire à l'arrêt de l'application s'ils sont utilisées de manière non sécurisées et réfléchies, il existe aussi

des fonctions qui sont devenues obsolètes après un certain temps, ces fonctions peuvent causer des failles de sécurité qui peuvent être facilement exploitable.

L'analyse statique automatique permet aussi de détecter des imperfections dans le code comme les parties du code mort ou bien les parties de code inatteignable, mais aussi certains problèmes de synchronisation liés au code source et bien d'autres encore.

4 L'Analyse dynamique :

a) Définition :

L'analyse dynamique (DAST, Dynamic Analysis Security Testing) ou bien le test de la boite noire (Black Box), est l'examen du programme lors de son exécution, il consiste à tester les fonctionnalités d'un logiciel avec un ensemble de valeurs en entrée et voir s'il répond au besoin fonctionnel et non fonctionnel, cette méthode ne peut pas prouver qu'un programme satisfait à cent pour cent une propriété donnée (impossible de réaliser un test avec toutes les valeurs possibles), mais elle peut détecter une violation de la propriété, en général ces tests sont réalisés après l'étape de l'implémentation d'un SDLC. (17)

b) Méthode de test dynamique :

Pour réaliser les tests dynamiques, il existe plusieurs méthodes comme le test unitaire, d'intégration, test de système, test de pénétration, mais aussi le test de robustesse.

c) Outil d'analyse dynamique :

L'analyse dynamique est effectuée de deux manières :

- L'analyse manuelle.
- L'analyse automatique.

Nous allons citer quelques outils d'analyse dynamique :

- JUnit,. Nmap, Burp.
- Metasploit est un outil open source pour les tests de pénétration (pentest),
- SQLmap est un outil open source utilisé dans les tests d'intrusions, son but est la détection automatique des failles de type SQL injection.

d) Comparaison test dynamique VS test statique :

Généralement dans les entreprises on utilise les deux méthodes, l'analyse statique est utilisée lors de la phase de codage et l'analyse dynamique est utilisée dans la phase de test, le tableau suivant montre quelques points comparatifs.

	Analyse statique	Analyse dynamique
Exécution du programme nécessaire	Non	Oui
Nécessiter d'avoir le code source	Oui	Non
Couvre toute l'application	Oui	Non
Tester la configuration	Non	Oui
Tester l'environnement de l'exécution	Non	Oui
Rapide	Plus rapide	Moins rapide
Détecte tous les bugs et les failles de sécurité	Non	Non
Utilisation des outils pour réaliser les tests	Oui	Oui
Les outils de test peuvent générer de faux positifs/négatifs	Oui	Oui
Détecter les failles de type : SQL injection, XSS et buffer over flow	Oui	Oui

Tableau 7: Comparaison entre l'analyse statique et l'analyse dynamique.

Dans le tableau 6, nous exposons quelques points de comparaison entre les deux méthodes de test statique et dynamique. On constate que les deux méthodes sont utilisées dans des phases différentes, l'analyse statique est utilisée lors de l'implémentation d'où la nécessité d'avoir le code source à portée de main, alors que l'analyse dynamique est utilisée lors de la phase de test et de vérification d'où la nécessité de l'exécution du programme. On constate aussi que ces deux méthodes ont des objectifs différents par exemple, la première ne peut pas tester la configuration du programme alors que la deuxième le fait, mais ces deux méthodes ont quelque point en

commun comme la détection des failles de type SQL injection et XSS et quelques autres vulnérabilités.

En conclusion, on constate que ces deux méthodes sont complémentaires et doivent être utilisées ensemble mais dans des phases différentes du cycle de développement pour assurer la sécurité d'un produit logiciel.

5 Autres méthodes de test :

Il existe d'autres types de tests pour assurer la sécurité des logiciels que nous allons citer :

- **Test de sécurité interactive des applications (IAST) :**
  Le test de sécurité interactive (IAST : Interactive Application Security Testing) est une combinaison entre le SAST et le DAST visant à tirer parti de la puissance et des avantages des deux méthodes, toutefois il est très difficile d'implémenter la méthode IAST, car elle nécessite la combinaison automatique des résultats obtenus lors des tests SAST et DAST, la combinaison des données manuelles ou semi-manuelles n'est pas vraiment une solution IAST.

- **Test de sécurité des applications mobiles (AST mobile) :**
  Mobile AST utilise une combinaison de SAST traditionnel et DAST, ainsi que l'analyse comportementale à l'aide de techniques statiques et dynamiques pour découvrir des actions malveillantes ou potentiellement risquées que l'application mobile peut subir (par exemple, l'activation du carnet d'adresses de l'utilisateur ou GPS).

6 Discussion :

Certaines vulnérabilités peuvent être trouvées uniquement avec les tests SAST(analyse statique), d'autres avec DAST (analyse dynamique). utilisés dans les deux sens rend le teste plus complet. De nombreuses applications web qui sont traditionnellement scannées avec les outils DAST utilisent également une quantité importante de code client sous la forme de JavaScript. Ce code doit également être analysé pour détecter les vulnérabilités de sécurité, généralement en utilisant l'analyse statique. Il existe d'autres raisons, mais la plus importante est que le test d'application avec une seule forme d'outil laisse un risque résiduel donc les applications les plus critiques doivent être testées à l'aide des techniques SAST et DAST. La bonne nouvelle est que plusieurs fournisseurs offrent les deux formes de test (par exemple IBM, HP ou encore Cigital comme il est montré dans le tableau 3 de la partie une du chapitre) par conséquent l'achat de deux outils distincts / services n'est plus nécessaire, de plus l'utilisation des deux méthodes ainsi que l'analyse manuelle diminue de façon considérable l'apparition de faux positifs même si cela reste encore non maitrisé complètement vu la difficulté qu'il y'a à trouver les faux positifs surtout

quand le système est complètement automatisé en d'autres termes, absence d'analyse manuelle (intervention humaine), cependant hormis la complémentarité, les différents outils qui utilisent une méthode par rapport à l'autre ne sont pas très efficace et ne permettent pas d'offrir un large choix en matière de résultat concret ou bien une analyse en un temps extrêmement réduit, pour cela la complémentarité des deux méthodes et leur utilisation synchronisé permet d'avoir des résultat beaucoup plus optimiste.

7 Conclusion :

Dans ce chapitre nous avons vu l'importance de la sécurité et plus particulièrement la sécurité applicative, la négligence de l'aspect sécuritaire en informatique peut engendrer des problèmes majeurs et peut causer des pertes faramineuses, ainsi plusieurs méthodes ont été développées afin de venir à bout des problèmes rencontrés, l'une d'elles est le SDL qui permet d'introduire la sécurité dans toutes les phases du cycle de développement, nous avons vu aussi la méthode d'analyse et de test dynamique qui est employée après l'exécution du programme et qui ne nécessite pas la lecture du code source, a l'inverse de la méthode d'analyse statique qui elle consiste à réaliser des tests directement sur le code source, ce type de test existe sous deux versions, l'analyse manuelle et l'analyse automatique, l'analyse statique automatique est très importante car elle permet de réduire le coût de développement en détectant les bugs dans l'étape d'implémentation (codage) là où la majorité des bugs se produisent et ainsi réduire l'effort effectué pour trouver ces erreurs en utilisant des outils automatiques. Dans l'ensemble, l'industrie des tests de sécurité semble mûrir et tend vers une meilleure maîtrise de la sécurité applicative.

Chapitre III : solution proposée

1 Introduction :

Notre mémoire de master s'est déroulée au sein de la filiale ELIT (**EL Djazair Information technology)** de groupe **Sonelgaz**.

Ce chapitre présente une étude exhaustive sur le système que nous allons compléter et les composants qui le constituent. Toutes les informations ont été rassemblées suite à des visites sur place et des entretiens avec les membres de l'équipe du département de sécurité des systèmes d'information, représentant aussi un portail vers la réalisation d'un système plus intuitif et plus pratique qui offrira une simplicité qui sera appréciée par les auditeurs qui utiliseront cette application.

Dans cette partie nous allons présenter notre organisme d'accueil ainsi que la description de leurs besoins.

2 Présentation de l'organisme d'accueil :

La filiale ELIT se charge principalement de l'activité du systèmes d'information, confiée à une direction générale au niveau de la Maison Mère du Groupe Sonelgaz, a été érigée en Société par actions, dénommée "EL Djazair Information Technology", par abréviation "ELIT Spa". Elle a été créée pour répondre à la stratégie du Groupe Sonelgaz de développer des moyens propres de maîtrise d'œuvre dans le domaine des Systèmes d'Information, et de disposer d'un pôle de compétences technologiques au service de ses sociétés, mais aussi de confier la propriété des Systèmes d'Information à une entité spécialisée et de focaliser les capacités de ses sociétés sur leurs métiers de base respectifs.

2.1 Rôles et attributions :

- Élaborer le schéma directeur SI du Groupe Sonelgaz.
- Assurer l'accès à l'information et aux applications et en garantir la sécurité, l'intégrité et la fiabilité.
- Mettre à la disposition du Groupe Sonelgaz les moyens informatiques et de télécommunications (ressources, matériels, infrastructures, etc.) nécessaires pour assurer le niveau de service attendu.
- Assurer des prestations en termes de besoins en systèmes d'information par la fourniture de services en mode client / fournisseur.
- Veiller au choix des normes, des standards et des méthodes, à des fins d'optimisation économique et technique et de faciliter l'interopérabilité et les échanges d'informations entre ces systèmes.

- Assurer le rôle de centre d'expertise du Groupe par le développement des ressources humaines et des méthodologies adaptées.
- Proposer, à terme, ces mêmes services aux clients externes.

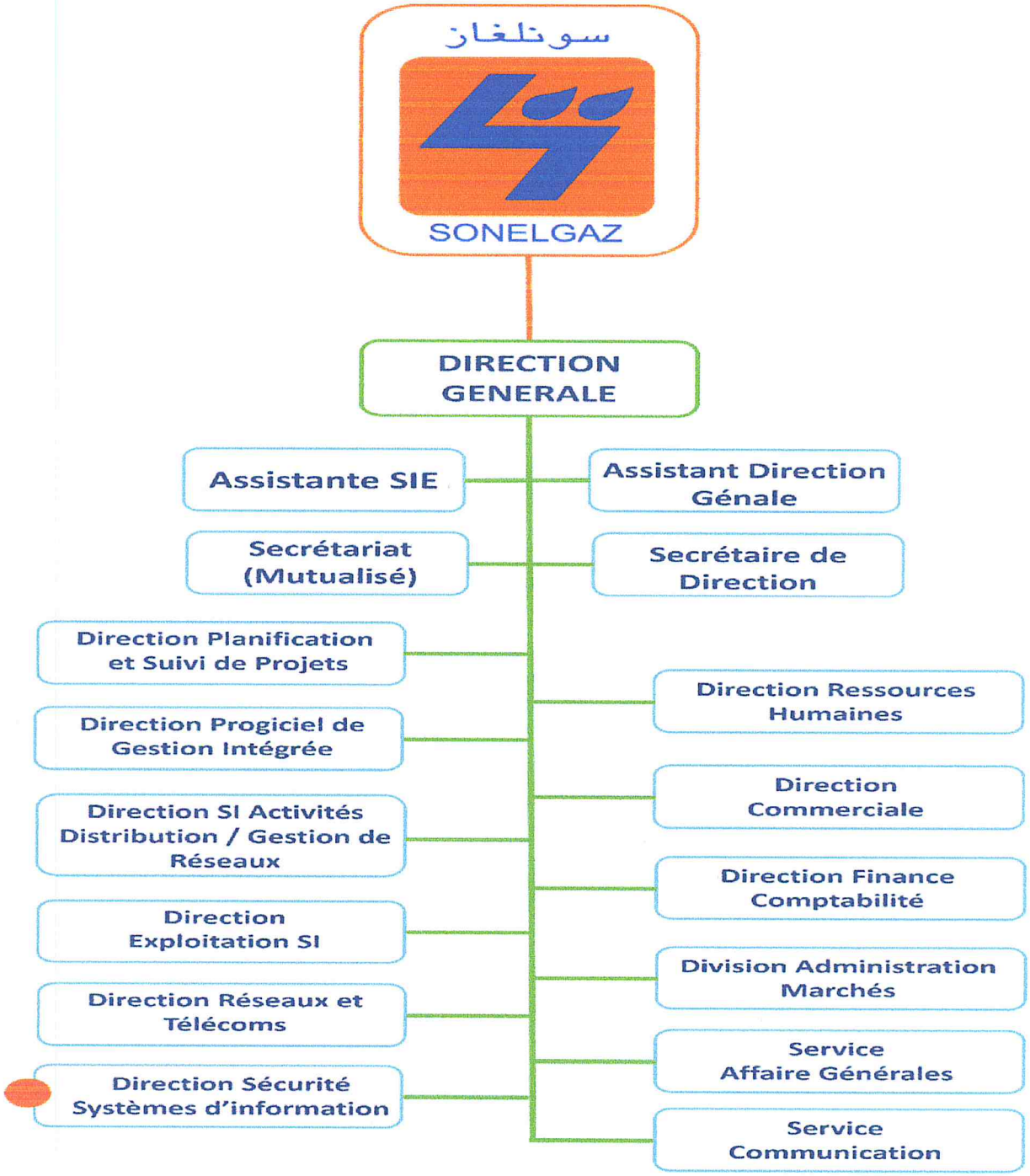


Figure 7: Organigramme de la filiale Elit filiale de sonelgaz.

À cet effet, ELIT est chargée notamment de :

a. *Au niveau stratégique :*

- L'élaboration de la politique des Systèmes d'Information et des technologies de l'information et de la communication du Groupe Sonelgaz ;
- La prise en charge des besoins des sociétés du Groupe Sonelgaz en matière d'informatique et de télécommunications.

b. *Quant au niveau opérationnel, ELIT s'emploie à :*

- Élaborer et mettre en œuvre les Systèmes d'Information destinés au pilotage et à la gestion des différentes activités des sociétés du Groupe Sonelgaz ;
- Mettre à la disposition des sociétés du Groupe Sonelgaz les moyens informatiques et de télécommunications (logiciels, matériels, infrastructures, etc.) nécessaires pour assurer le niveau de service attendu ;
- Assurer la maintenance et l'administration des Systèmes d'Information, des plates-formes et des équipements mis à la disposition des utilisateurs ;
- Assurer l'accès à l'information et aux applications et en garantir la sécurité, l'intégrité et la fiabilité ;

3 La mission du département de la sécurité des systèmes d'information :

La Direction Sécurité des Systèmes d'Information est chargée de la sécurité du patrimoine informationnel, des systèmes d'information du Groupe Sonelgaz et de garantir l'intégrité, la confidentialité, la disponibilité et la traçabilité des données de l'ensemble des systèmes d'information de la société.

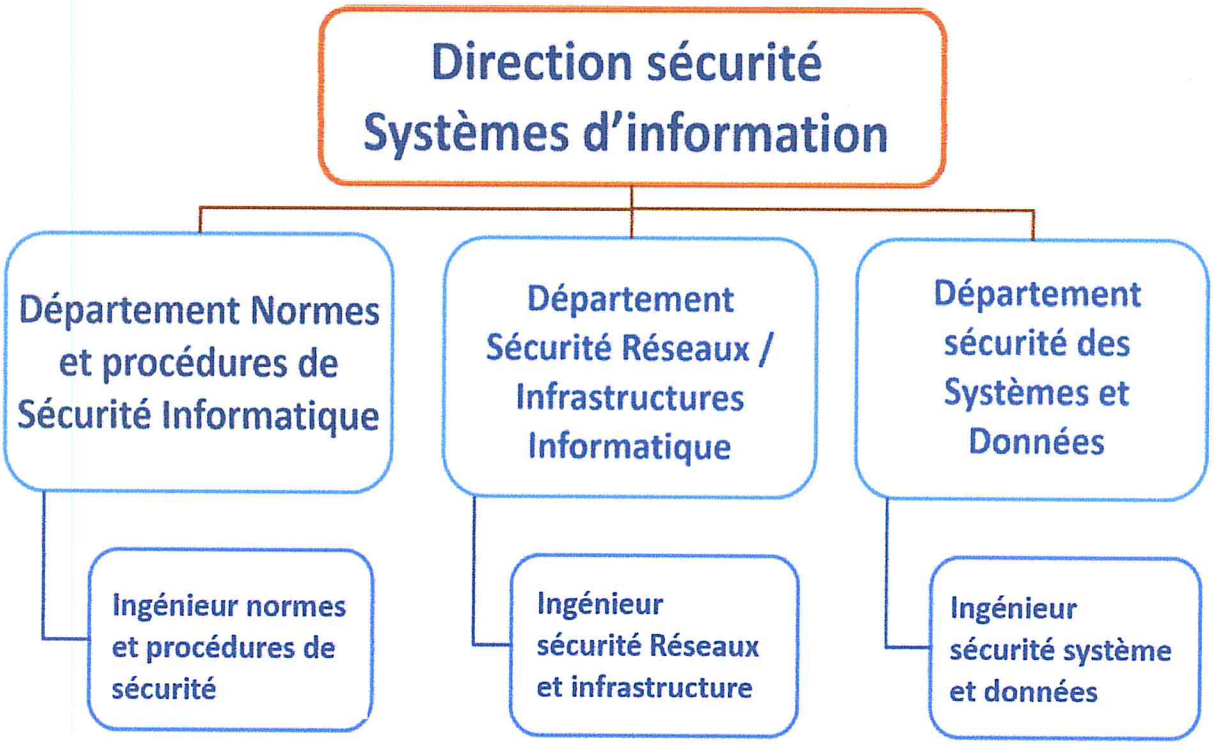


Figure 8: Organigramme du département de sécurité des systèmes d'information.

❖ **Rôles et attributions :**

- Définir la stratégie sécurité des systèmes d'information du Groupe Sonelgaz.
- Mettre en place la politique de sécurité du Groupe Sonelgaz et veiller à son application.
- Sensibiliser les différents acteurs des systèmes d'information sur la sécurité des systèmes d'information.
- Réaliser des audits de sécurité des systèmes d'information (application et contrôle des plans de sécurité préétablis).
- Identifier les risques informatiques pesant sur le Groupe Sonelgaz et leurs éventuelles conséquences.
- Assurer la veille technologique dans le domaine de la sécurité informatique.

4 Étude du système existant :

Dans cette partie nous allons présenter le système actuel, ses point forts et ses points faibles.

4.1 Introduction :

Dans cette partie nous allons nous intéresser au système existant afin de déterminer ces points forts et ces points faibles. En fonction de cette analyse et en fonction de la critique de l'existant, on peut déceler le fond du problème et trouver une solution pour améliorer le système existant.

L'audit code source manuel est une manière insuffisante de détecter les vulnérabilités surtout dans des entreprises qui misent en premier lieu sur deux facteurs principaux qui sont le temps et le coût, cette technique est exhaustive et non fiable, de plus, des études ont montré que l'audit dynamique ne suffit pas à déceler toutes les failles de sécurité que rencontre un projet lors de sa phase d'audit.

Cependant, l'approche de l'analyse statique du code source présente plusieurs imperfections, deux d'entre elles sont les faux négatifs et les faux positifs qui faussent parfois les résultats de l'analyseur, mais aussi, remettent en cause toute la phase d'audit.

4.2 Les besoins de l'entreprise :

L'entreprise dans lequel nous effectuons notre stage a exprimé plusieurs besoins et sont énumérés ci-dessous :

1. Minimiser le temps d'audit du code source et le temps de scan.
2. Détecter les failles de sécurité de type injection présente dans le code source.
3. Développer une application facile et intuitive qui permet d'aider les auditeurs à accomplir la phase d'audit du projet.
4. Mettre en place un système de scan qui supporte en premier lieu le langage PHP et qui peut par la suite s'étendre vers d'autres langages.
5. Générer des rapports d'audits et permettre leur archivage et leur gestion par l'auditeur.
6. Établir une interaction efficace et robuste entre le système proposé et le système actuel.
7. Diminution du taux de faux positifs et des faux négatifs.

4.3 Exposition du système actuel :

La filiale ÉLIT a pour objectif d'auditer et de tester la sécurité des produits logiciel développé par le groupe Sonelgaz, mais aussi d'autres projets qui appartiennent à des organismes indépendants (clients). Le système actuel est une application web de gestion des demandes d'audit qui possède un ensemble de fonctionnalités afin de réaliser l'audit et garantir un suivi complet jusqu'à la mise en production de l'application ou son exploitation, les principaux acteurs sont :

- Les clients, qu'ils soient internes ou externes au groupe sonelgaz, doivent passer par les mêmes étapes afin de gérer de façon automatique les demandes d'audit.
- Le responsable d'audit a plusieurs missions comme le traitement des requêtes d'audit effectué par le client ainsi que l'affectation des projets a des équipes d'audit.

- L'auditeur audit les projets et traite les données qui en résultent.

Pour l'instant, ce système ne gère que les informations récoltées par l'auditeur lors de l'analyse dynamique (test de pénétration, test de configuration…etc.) du projet, de plus il facilite la communication entre l'auditeur et le client, en premier lieu ce dernier doit remplir une demande contenant un ensemble d'information sur le projet à auditer, cette demande comporte :

1. Les informations concernant l'interlocuteur (entreprises).
2. Les adresses des serveurs qui sont nécessaires au bon fonctionnement du projet ainsi que leurs rôles (serveur d'application, serveur de fichier, serveur de base de données...etc.).
3. Les informations requises pour l'accès aux différents serveurs (nom d'utilisateur, mot de passe, clés SSH...etc.).
4. Une description précise des exigences et des directives du client concernant l'audit réalisé sur son projet.

Après avoir reçu ces informations de la part du client, le responsable d'audit effectue plusieurs opérations :

1. Vérifications de la véracité des données entrées par le client lors de la demande.
2. Si les données n'autorisent pas l'accès, le responsable envoie une demande de correction au client.
3. Le responsable d'audit affecte des projets à un ou plusieurs auditeurs pour effectuer l'audit dynamique.
4. L'auditeur rédige par la suite un rapport contenant toutes les insuffisances et les incomplétudes détectées lors de la phase d'audit et l'envoi au client(développeur).
5. Après la correction des failles détectées, le client ré effectue une nouvelle demande d'audit qui passera par les mêmes phases.
6. Un rapport final est généré et le responsable d'audit envoie ce rapport au client pour la mise en production du projet.

Le diagramme illustré dans la page suivante (figure 10) montre en détail les différentes phases d'audit et l'interaction entre les acteurs du système et le système en lui-même.

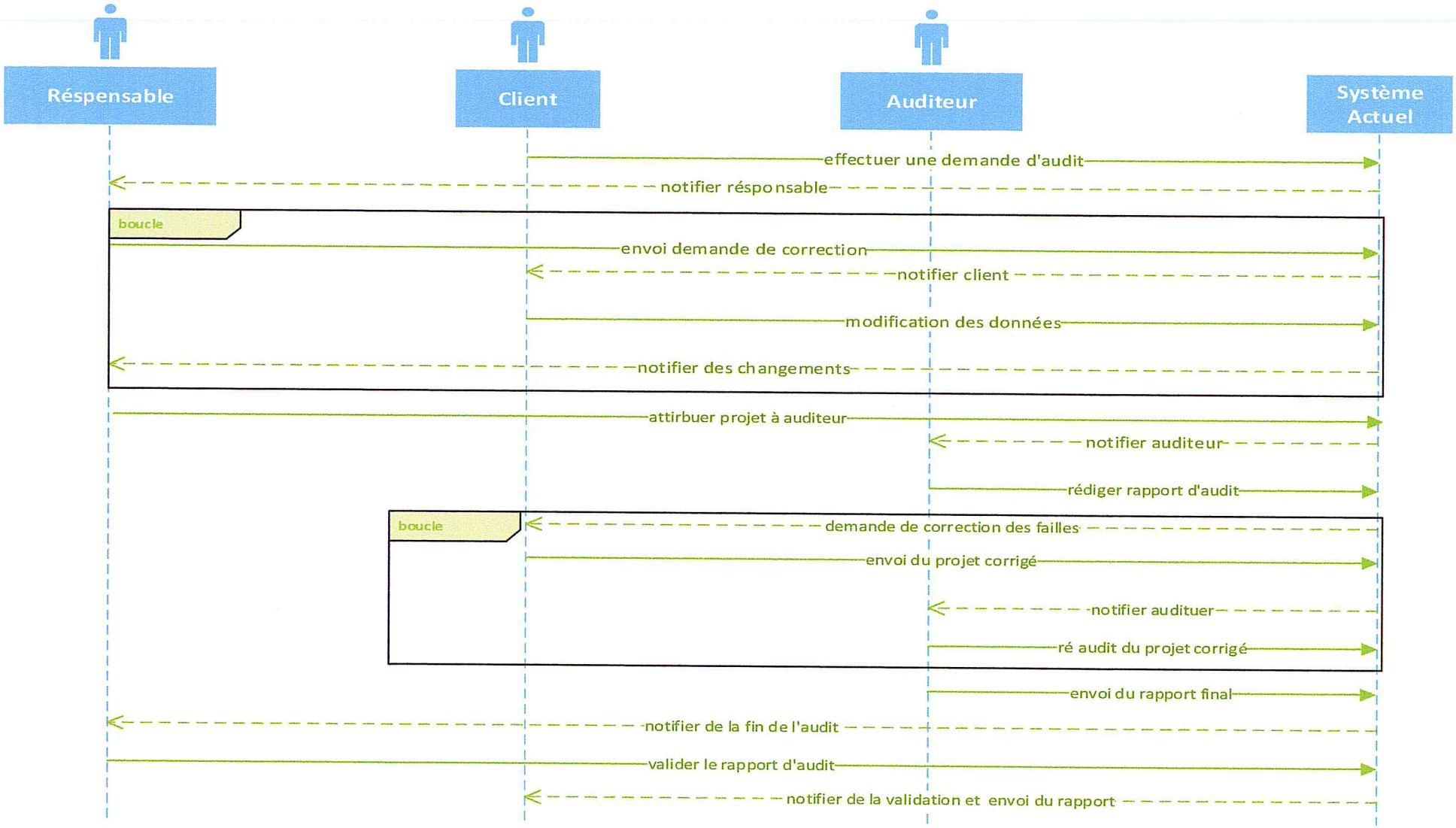


Figure 9: Diagramme de séquence des opérations du système actuel.

5 Critique de l'existant :

Le système actuel qu'ELIT a développé pour mener à bien leur mission d'audit comprend entre autres des points forts et des points faibles que nous allons énumérer ci-dessous :

5.1 Les points forts du système :

- Dispose d'une plate-forme qui permet un échange d'information nécessaire afin d'effectuer l'audit du produit développé par le client.
- Une interface auditeur simple qui permettra de consulter tous les projets qui lui ont été affectés ainsi que les rapports d'audit déjà archivés.
- Un espace réservé aux clients qui permet de faire des demandes d'audit, mais aussi la mise en place d'un formulaire qui permettra d'exprimer la demande du client ainsi que toutes les informations nécessaires à l'accès (droits d'accès aux serveurs, délai d'audit, etc.)
- La classification automatisée des vulnérabilités détectées ainsi que l'archivage des rapports.

5.2 Les points à déplorer du système :

- Cette plate-forme est une application web qui sert seulement à automatiser la gestion des demandes d'audit.
- Aucun outil d'audit (tests de pénétration) propre au système n'existe dans le système actuel
- Le système ne dispose d'aucun analyseur d'audit code source (statique) que ce soit du côté de l'analyse du code ou bien des résultats rédigés dans le rapport final.

Par ailleurs, nous devons savoir que le nouveau système conçu n'améliora pas le système actuel, mais le complétera d'un point de vue fonctionnel et opérationnel.

6 Exposition du système proposé :

Afin de répondre aux besoins exposés par l'entreprise nous avons élaboré un système qui à la fois apporte une complémentarité aux insuffisances du système existant énumérées plus haut, mais aussi offre de nouvelles perspectives pour un meilleur audit et de meilleurs résultats.

Ce système doit offrir une simplification lors de la phase d'audit, une réactivité dans le traitement, une analyse rapide et efficace, mais aussi la détection des erreurs et des vulnérabilités contenues dans le code source du programme que l'auditeur doit analyser.

Tout d'abord, avant qu'il ne consiste à être un outil d'audit code source, notre projet est en premier lieu un système à part entière qui a pour but de faire de l'audit à l'échelle code source, il se détermine par deux aspects :

- ✓ L'aspect fonctionnel :
  - Le système reçoit les résultats lors de sa phase d'analyse et rassemble toutes les informations de façon claire et organisée.
  - Le système exposera à l'auditeur toutes les vulnérabilités détectées dans le code source et lui affichera la partie du code ou la menace a été détectée dans le but de vérifier si vraiment la menace en est une ou bien c'est un faux positif.
  - Le système permet à l'auditeur de supprimer une vulnérabilité s'il estime que la menace détectée n'est pas dangereuse et n'aura aucun impact.
  - Le système fournit à l'auditeur un moyen d'ajouter une vulnérabilité si au court de l'audit manuel sur une portion de code il découvre une vulnérabilité que le système n'a pas réussi à détecter, ce qui est connu par les faux négatifs.
  - Le système offre toujours la possibilité à l'auditeur de faire de *l'audit manuel* sur la portion de code où les vulnérabilités ont été détectées.
- ✓ L'aspect opérationnel :
  - L'analyseur se base en particulier sur les « input » qu'un utilisateur peut insérer, et en partant de ce principe on alimente une librairie qui contient des fonctions dangereuses et obsolètes ainsi que les vulnérabilités les plus répondues, pour qu'on puisse comparer et dire si la menace(faille) trouvée dans le code en est vraiment une ou bien ce n'est qu'une fausse alerte.
  - L'analyseur proposera dans l'aspect opérationnel une partie remédiation aux résultats acquise lors de la phase de scan

6.1 Description du système proposé :

Dans cette partie nous allons voir en premier lieu le schéma global des deux systèmes ainsi que leur interaction, ensuite, nous verrons la conception détaillée du système proposé.

La figure 11 ci-dessous est une représentation globale de l'interaction entre le système qu'on propose et le système actuel, chaque système se compose de plusieurs modules qui communiquent entre eux à travers une base de données.

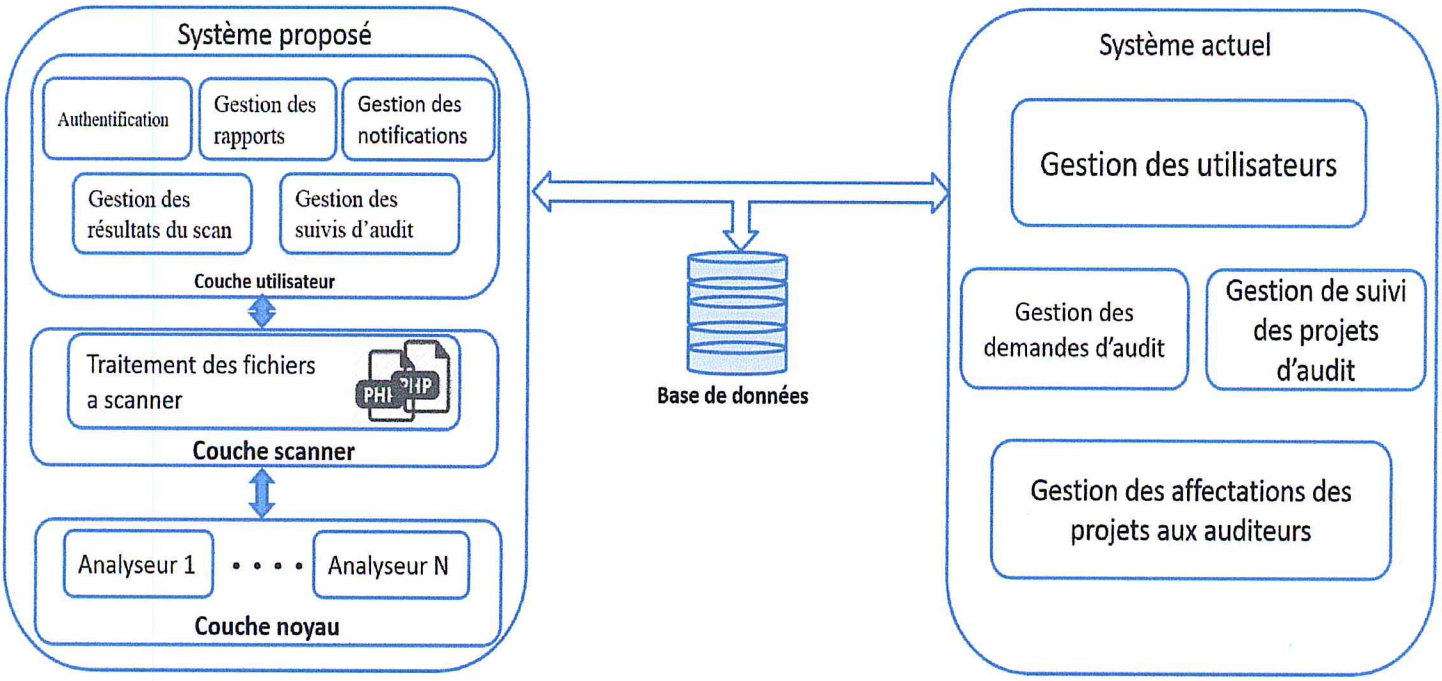


Figure 10: schéma global de l'interaction des deux systèmes.

Le système proposé se compose de 3 couches essentielles à son bon fonctionnement et qui permettent d'avoir à la fois des résultats satisfaisants en matière de détection de vulnérabilités et une simplicité dans l'acquisition de ses informations.

a. **Couche utilisateur :** la 1ère couche représente la couche utilisateur elle offre l'une des principales fonctionnalités de notre système et qui est la gestion des résultats obtenus par l'auditeur la présence de ce dernier est le dernier maillon de la chaîne avant la validation finale du résultat, elle se compose de six modules bien distincts :

- Le module d'*Authentification* est un système de session réservé à nos deux types d'utilisateurs, le responsable et les auditeurs, à l'aide de ces sessions nous instaurons une politique d'accès qui permettra de faire en sorte que seuls l'auditeur et le responsable puissent y accéder.
- Le module de *contrôle d'entrées utilisateur* permet entres autres de vérifier si l'utilisateur a déjà scanné un projet, si oui, un contrôle doit être mis en place pour qu'il n'y ait pas un écrasement répété des données récoltées et stockées dans la base de données lors l'analyse statique du code source (scanne).
- Le module de *Gestion des rapports d'audit* se charge de traiter et d'afficher les informations qui ont été récupérées de la couche scanner.

  De plus, l'auditeur a grâce à ce module a la possibilité de supprimer définitivement les informations (vulnérabilités) que l'analyseur détecte et qui s'avèrent après l'audit être

de fausses alertes (**faux positifs**), mais aussi a la possibilité d'ajouter manuellement une vulnérabilité que l'analyseur n'a pas pu détecter et qu'on appelle un (**faux négatif)** (voir chapitre II) et qui sera ajoutée au rapport final après que la phase d'audit se soit terminée.

- Le module de *Gestions des notifications* permet une communication bidirectionnelle entre les deux systèmes, en effet grâce à ce module le responsable notifie en temps réel les auditeurs sur les nouveaux projets auxquels ils sont affectés et cela à travers le système existant, de plus le système proposé sera en mesure de notifier le responsable dès qu'un projet en phase d'audit change de statut.
- Le module de *Gestion des suivis d'audit* est l'un des modules les plus importants dans cette première couche, il fournit des informations primordiales à propos de l'état d'avancement de l'audit ainsi que les différentes phases du projet par lequel il passe et que nous allons énumérer ci-dessous :
    1. *Phase d'audit* : c'est la phase qui vient tout juste après l'affectation du projet a l'auditeur afin que ce dernier commence à analyser le projet(audit).
    2. *Phase de ré audit* : c'est la phase ou l'auditeur doit ré analyser le projet après que ce dernier ait été envoyé aux développeurs pour qu'ils corrigent les erreurs détectées lors de la phase précédente.
    3. *Phase de validation :* c'est la dernière phase du suivi d'audit et qui comme son nom l'indique valide les résultats ainsi que le projet tout entier pour qu'ensuite un rapport final soit généré et envoyé aux clients.
- Le module de *gestion des résultats du scan* représente le portail qui permet la communication entre la couche utilisateur et la couche scanner, il récupère les données envoyées par la deuxième couche (voir couche scanner) afin qu'il les traites et les affiches que ce soit par catégories, par nombres de failles détectées ou bien par les vulnérabilités appartenant à un fichier bien précis du répertoire du projet.

Toujours est-t-il que malgré toutes les règles de remédiation et les fonctionnalités d'analyse qu'offre le système, la présence et l'implication de l'expert (auditeur) est obligatoire pour valider les résultats et ainsi éviter toute validation de faux positifs et de faux négatifs dans le rapport final.

Pour finir, on peut dire que la couche utilisateur est une porte d'accès aux couches suivantes, mais aussi une plate-forme d'affichage et de modifications des résultats obtenus lors de l'analyse du code source.

b. **Couche scanner :** d'après la description qu'on vient de voir dans la couche utilisateur on conclut que le rôle principal de la 1ére couche est de gérer et stocker les données des utilisateurs dans une base de données.

La couche scanner a pour but de piloter la couche 3 (noyau), elle effectue le traitement des données avant et après l'analyse du code source pour le noyau, à cet effet la 2e couche se compose de deux modules dépendants l'un de l'autre et que nous allons présenter ci-dessous :

1. *Prés analyse* : ce module traite deux principales informations qui sont récupérées depuis la 1re couche (couche utilisateur).

   Premièrement, le langage sélectionné, il va déterminer à quel analyseur du noyau (couche 3) ce module va faire appel.

   Deuxièmement, le chemin du répertoire du projet à scanner, il va être utilisé pour générer une liste contenant le chemin des fichiers qui se trouvent dans ce répertoire, on note que les fichiers à analyser concernent le langage qui a été choisi par l'auditeur, en d'autres termes si le répertoire contient des fichiers d'un autre langage, ces derniers ne seront pas pris en charge par l'analyseur et une erreur sera affichée pour que l'auditeur y remédie.

2. *Post analyse* : ce module recevra les résultats de l'analyse et va réaliser les opérations suivantes :

   - Traitement des résultats afin de les renvoyer à la couche utilisateur et cela dans le but de les afficher d'une manière simple et compréhensible.
   - Récupérer les résultats obtenus lors de la phase de l'analyse statique et les stocker dans la base de données réservée à cet effet.

c. **Couche noyau :** Cette couche se charge principalement de l'analyse statique du code source, elle peut contenir plusieurs analyseurs (outils), chaque outil correspond à un langage donné, pour chacun de ces outils plusieurs techniques sont utilisées et implémentées, ces techniques et leurs implémentations peuvent changer d'un outil a un autre selon le type du langage utilisé (on verra par la suite la conception d'un analyseur qui est spécifique au langage PHP). Cependant le résultat des différents analyseurs doit suivre la même structure pour garantir la bonne gestion des informations qui sont envoyées à la couche 2, cela garantie aussi un ajout facile de plusieurs autres analyseurs dans la couche 3 (noyau), rendant le cœur du système extensible et robuste afin de gérer le maximum de langages et de traiter le maximum de projets possible lors de la phase d'audit sans avoir à changer le fonctionnement de la couche scanner.

7 Interaction entre les deux systèmes :

Le diagramme de séquence qui suit (figure 12) montre comment et à quel moment les deux systèmes interagissent entre eux et décrit précisément les étapes de l'opération d'audit durant l'utilisation du nouveau système et la façon avec laquelle les informations (affectation de sujets, rapport d'audit...etc.) sont envoyées et échangées à travers les deux systèmes…

Après que le client ait fait sa demande d'audit, il l'envoie pour que le responsable la traite, ensuite, chaque acteur devra suivre une marche à suivre (scénario nominal) afin de mener à bien le processus d'audit.

Scénario 1 : Interaction entre le système actuel et le système proposé.	
Objectif : montrer l'interaction des systèmes et leurs principales opérations.	
Acteurs :Auditeur, responsable, client.	
Précondition : la demande d'audit doit être déjà effectuée par le client.	
Post condition : une génération du rapport final de l'audit et son envoi au client.	
Scénario nominal : 1 : le responsable affecte un projet a un auditeur 2 : le système actuel envoi notification au système proposé. 3 : le système proposé notifie l'auditeur. 4 : l'auditeur sélectionne le projet a audité. 5 : le système proposé analyse le projet. 6 : Affichage information générée par le système proposé. 7 : l'auditeur met à jour les informations. 8 : le système notifie de la réussite de la mise à jour.	9 : l'auditeur valide le rapport d'audit statique 10 : L'auditeur rédige rapport d'audit dynamique 11 : le système actuel envoie les deux rapports 12 : le client renvoie le projet corrigé 13 : le système actuel notifie de la réception du projet. 14 : l'auditeur valide le projet 15 : le responsable valide les deux rapports d'audit 16 : le système actuel notifie de la fin d'audit 17 : le système actuel notifie de la validation et de l'envoi des deux rapports.

Tableau 8: scénario nominal de notre diagramme de séquence.

Le digramme de séquence découlant est alors le suivant (figure12) :

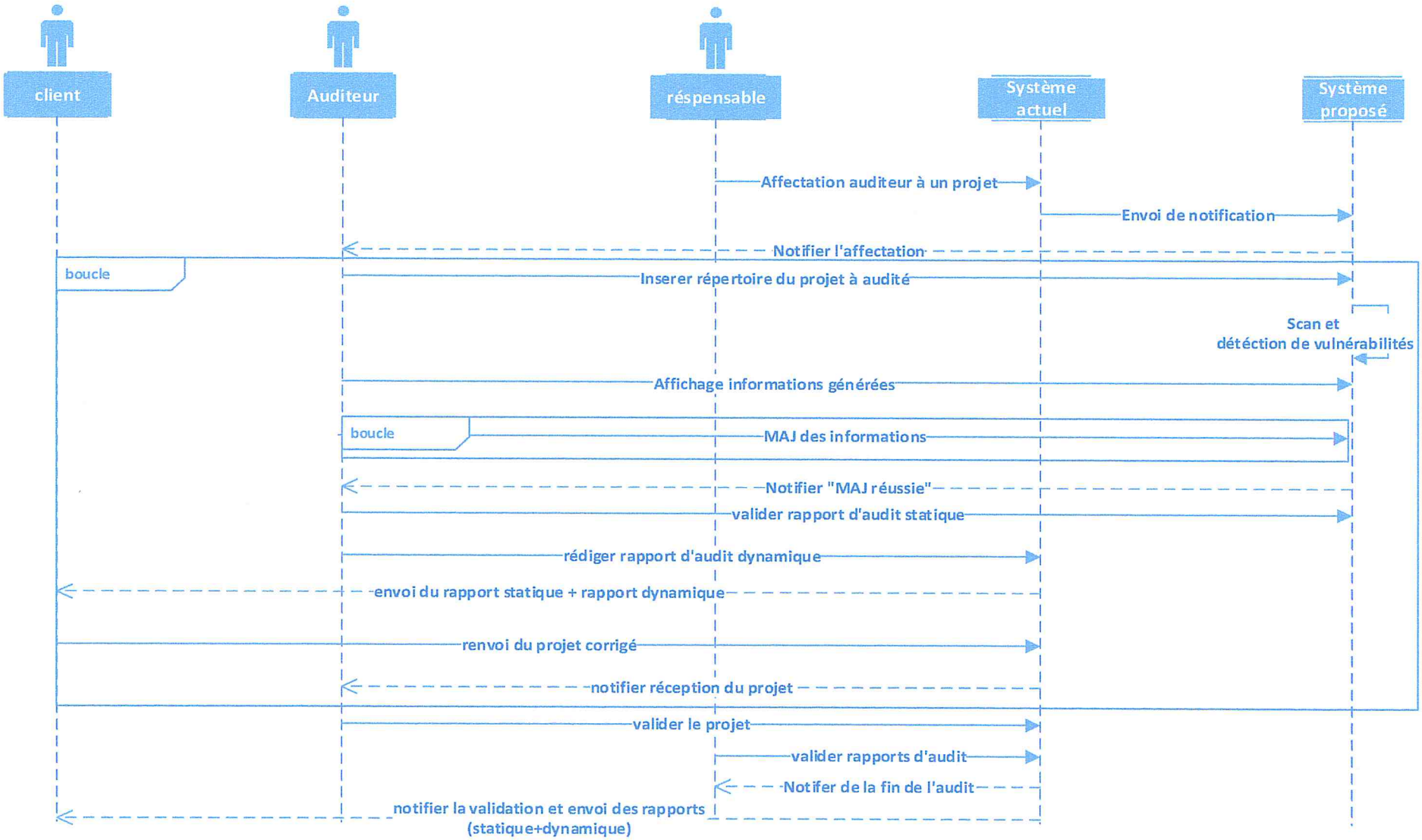


Figure 11 : diagramme de séquence montrant l'interaction entre les deux systèmes.

8 Conception du système proposé :

Le système offre nombreuses fonctionnalités, en effectuant toutes ces opérations nous pourrons considérer que la phase d'audit code source a été accompli de façon rigoureuse.

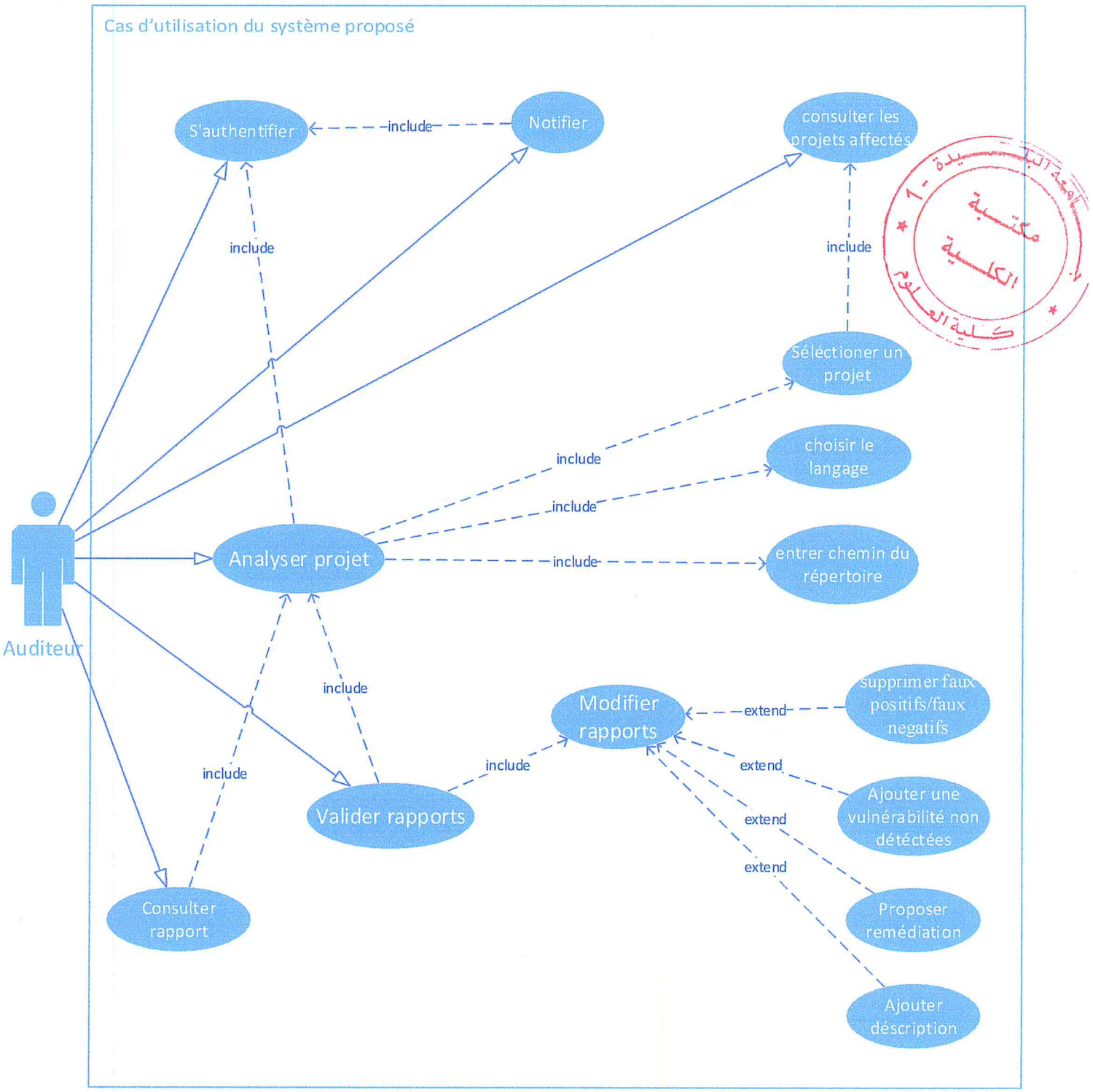


Figure 12: Cas d'utilisation du système proposé.

Nous allons en premier lieu mettre en place une fiche descriptive (tableau 9) d'un exemple réel du fonctionnement de notre système afin de définir notre cas d'utilisation (figures 13) :

Cas n° 1
Nom : Faire l'analyse statique du code source (auditer) Acteur(s) : Auditeurs Description : Un ou plusieurs auditeurs peuvent auditer un projet selon le choix et l'affectation du responsable. Auteur : auditeur (adjailia romaissa) Date(s) : 10/05/2017 (première rédaction) Préconditions : L'utilisateur doit être authentifié en tant qu'auditeur (Cas d'utilisation « S'authentifier ») Démarrage : L'utilisateur a demandé la page « Consultation des projets à affecter »

Tableau 9: fiche descriptive de notre cas d'utilisation.

Ensuite, nous allons décrire les scénarios qui explicitent la chronologie des actions qui seront réalisées par l'utilisateur (tableau 10), il existe 3 parties :

- ✓ **Le scénario nominal :**

  Il s'agit ici de décrire le déroulement idéal des actions, où tout va pour le mieux.

Le scénario nominal
1. Le système affiche une page contenant la liste des projets qui ont été affectés à l'auditeur. 2. L'utilisateur consulte les rapports et sélectionne un projet à auditer. 3. Le système affiche une nouvelle page qui permettra d'analyser le projet sélectionné. 4. L'utilisateur doit obligatoirement choisir le langage et entrer le chemin du répertoire. 5. Le système génère un rapport contenant toutes les vulnérabilités détectées. 6. L'utilisateur consulte le rapport et procède à une modification de ce dernier. (Suppression, ajout, description, remédiation 7. L'utilisateur peut ensuite consulter une nouvelle fois le rapport et le validera définitivement. 8. Le système sauvegarde les changements et valide le projet en modifiant son statut.(projet audité)

Tableau 10: scénario nominal de notre cas d'utilisation.

- ✓ **Le scénario alternatif :**

  Ici, il s'agit de décrire les éventuelles étapes différentes liées aux choix de l'utilisateur, par exemple. C'est le cas des étapes liées à des conditions.

Le scénario alternatif
2.a *l'utilisateur* décide de quitter la consultation des projets qui lui sont affectés.
2.b *l'utilisateur* décide de ne sélectionner aucun projet à auditer, il ne fait que consulter.
5.a *l'utilisateur* décider de quitter la consultation du premier rapport généré.
6.a *l'utilisateur* décide de ne faire aucune modification sur le rapport.
7.a *l'utilisateur* décide de valider le rapport sans consulter une nouvelle fois.
7.b *l'utilisateur* décide de quitter la consultation du rapport sans valider.

Tableau 11: scénario alternatif de notre cas d'utilisation.

✓ **La fin et les post-conditions :**

- La fin permet de récapituler toutes les situations d'arrêt du cas d'utilisation.
- Les post-conditions nous indiquent un résultat tangible qui est vérifiable après l'arrêt du cas d'utilisation et qui pourra témoigner du bon fonctionnement.

Fin : Scénario nominal : aux étapes 2, 5 ou 7, sur décision de l'utilisateur.
Post-conditions : Scénario nominal : aux étapes 6, 8.

Tableau 12: la fin et les postes conditions de notre cas d'utilisation.

8.1 Description de la base de données :

Le point principal qui relie les deux systèmes est justement la base de données, elle sera utilisée pour garantir l'échange d'informations, chaque module du système aura besoin d'informations qui ont initialement été entrées dans un module n'appartenant pas au même système.

La base de données comme montrée ci-dessous (figure 14) illustre avec un code couleurs, les tables appartenant au système initial et les tables ajoutées pour enregistrés les informations générées par le système d'audit code source.

8.1.1 Rôle et description de quelques tables :

- ***Table suivie audit :*** cette table sert à stocker les informations nécessaires au suivi du projet qu'on audit, elle prend en compte le statut du projet relatif à la phase d'audit dynamique qui s'effectue dans le système existant(actuel), mais aussi le statut résultant du suivi d'audit statique. Par conséquent elle permet de synchroniser la validation du projet dans sa globalité en prenant en compte les deux méthodes d'audit afin que le rapport soit prêt à être envoyé au client.
- ***Table suivie d'audit code source :*** la table suivie est propre au système proposé, elle a été ajoutée afin de stocker seulement les informations du projet qui est analysé par le

système d'audit code source, ces données seront récupérées à travers la clé étrangère « *id demande* » stockée dans la table « suivi audit ».

- ***Table Notification :*** cette table est celle qui relie nos deux systèmes elle leur permet de de stocker des informations afin que nos différents acteurs puissent communiquer entre eux en envoyant des notifications.
- ***Tables failles :*** cette table concerne premièrement les vulnérabilités détectées, on y stocke le nom de la vulnérabilité, numéro de la ligne, et la ligne de code ou a été détecté la vulnérabilité. Deuxièmement, elle stocke aussi les informations de prévention ce qui veut dire que si le système détecte une fonction dangereuse ou obsolète cette même table devra contenir le nom du « warning » ainsi que les mêmes informations citées premièrement, ces deux failles seront différenciées par l'attribut « type_faille ».
- ***Table demande d'audit :*** cette table est le point de départ par lequel passe le système actuel, les informations entrées par le client (nom du projet, date, descriptions, etc.) sont traitées et enregistrer dans la table demande audit afin que ces demandes soient consultées par le responsable dans le but de les affecter aux auditeurs.

Les tables du système proposé stockent essentiellement les informations générées lors de l'utilisation de l'analyseur statique de code source, le système actuel utilisera ces informations pour terminer le rapport final qui a été déjà rédigé partiellement par l'auditeur au moment de la phase d'audit dynamique (test de pénétration, test de politique de sécurité, etc.).

8.1.2 Politique d'accès à la base de données :

Comme les deux systèmes interagissent avec la même base de données, un risque de conflit d'accès existe, pour remédier à cela nous avons mis en place une politique d'accès que nous allons définir ci-dessous :

- ✓ Le système actuel peut accéder aux tables réservées à stocker les informations générées par le système proposé seulement en mode lecture.
- ✓ Le système proposé peut accéder aux tables réservées à stocker les informations enregistrées par le système actuel seulement en mode lecture
- ✓ Les deux systèmes ont le droit d'accéder en mode lecture et écriture seulement dans la table notification (figure 14).

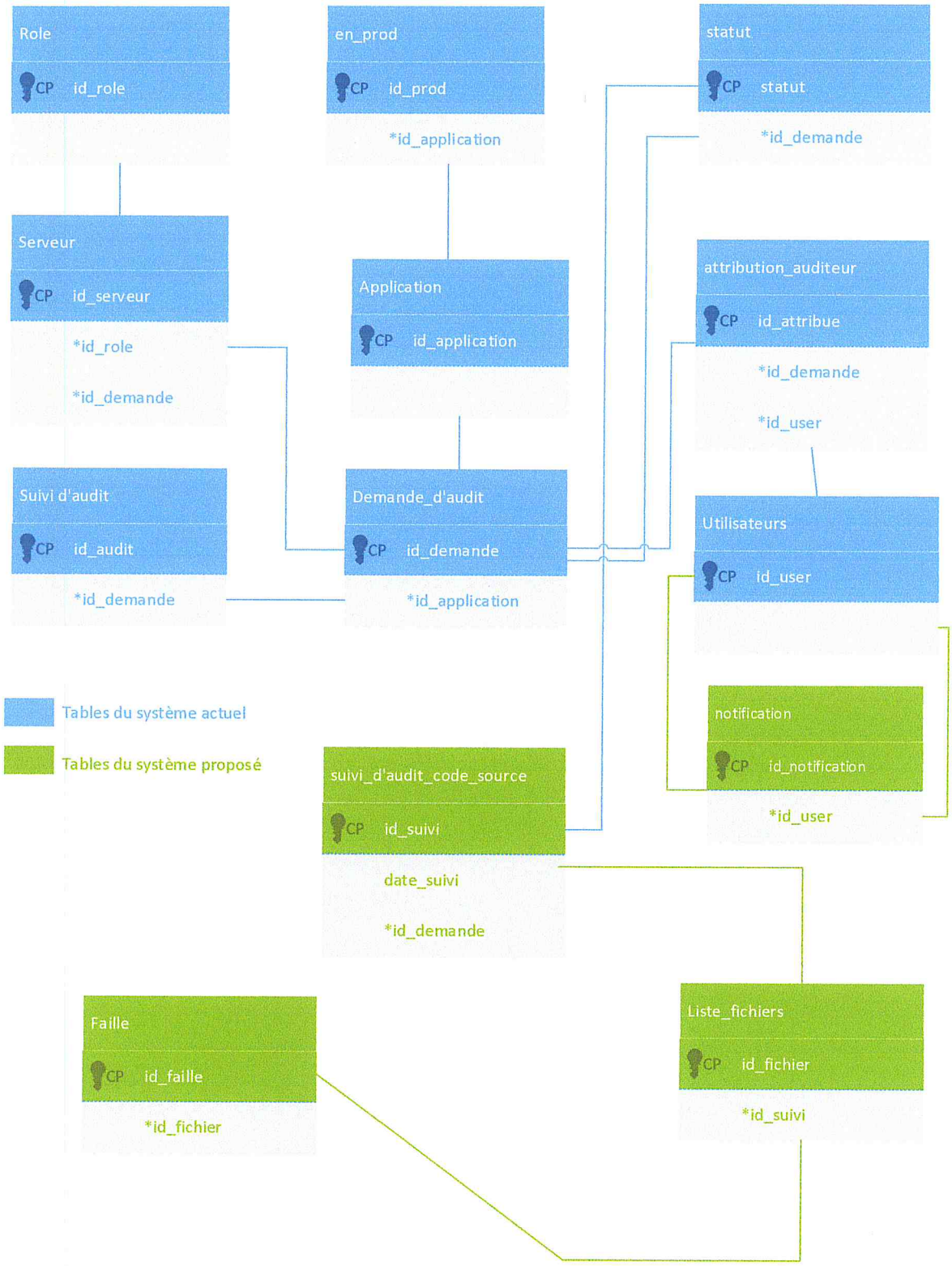


Figure 13: Diagramme de base de données.

8.1.3 **Identification des tables :**

Ci-dessous le les attributs que contiennent les tables de la base de données de notre système ainsi que leurs désignations et leurs types.

Tables	Attributs	Désignation	Type
Notification	Id_notif	Identificateur de la notification	Entier
	from	Identifiant de l'envoyeur	Entier
	to	identifiant du récepteur	Entier
	Vue	Vue	Booléen
	Msg	Message	Chaîne de caractère
	Id_système	Identificateur du système	Entier
Suivi_audit_code _source	Id_statut_d	Identifiant du statut dynamique (table de système actuel)	Entier
	Id_suivi audit	Identifiant du suivi audit code source	Entier
	lang	Langage	Chaîne de caractère
	Id_suivi*	Identifiant suivi (de la table suivie)	Entier
	Statut	Statut	Chaîne de caractère
Faille	Id_faille	Identifiant de la faille	Entier
	Id_suivi	Identifiant suivi	Entier
	Nom_vul	Nom vulnérabilité	Chaîne de caractère
	Nom_var	Nom variable détetectée	Chaîne de caractère
	Num_ligne	Numéro de la ligne	Entier
	Code_ligne	Code source de la ligne	Chaîne de caractère
	Id_fichier	Identifiant du fichier	Entier
	Type_vul	Type de la vulnérabilité	Chaîne de caractère
	Descrip	Description de la vulnérabilité	Chaîne de caractère
	Remédiation	Remédiation de la vulnérabilité	Chaîne de caractère
Liste fichiers	Id_fichier	Identifiant du fichier	Entier
	Id_suivis	Identifiant du suivi d'audit	Entier
	Chemin_rap	Chemin du répertoire du projet	Chaîne de caractère

Tableau 13: attributs et dictionnaires de données des tables.

8.2 Conception d'un analyseur PHP :

Nous avons dit que la 3e couche (noyau) peut contenir plusieurs analyseurs, chacun de ces outils est conçu pour un langage donné pour cela différentes techniques peuvent être utilisés, on va voir ici comment nous avons conçu un analyseur dédié au langage PHP (notre outil ne prend pas en charge PHP comme langage orienté objet). Selon W3 Techs, PHP est le langage de programmation côté serveur le plus couramment utilisé, avec 82,6% des serveurs web déployer.

Les applications développées avec ce langage en tendance à être destiné à un grand public via internet, cela expose ces applications à un risque d'utilisation malveillante. Si les développeurs ne prennent pas en considération les risques venant de l'extérieur (inputs), un attaquant peut prendre le contrôle de toute l'application au pire des cas via des injections. Notre analyseur a comme objectif principal est de détecter tous les points d'entrer des utilisateurs (input) et de les tracer pour voir si l'utilisation de ces données peuvent mener à des injections ou non, l'outil développé doit pouvoir en 1er lieu détecter les vulnérabilités de type XSS, SQL INJECTION, PATH TRAVERSALLE, FILE INPUT et quelques autres types d'injection, mais aussi il doit détecter l'utilisation des fonctions dangereuse ou obsolète existante dans le langage PHP, la figure suivante illustre la structure de notre analyseur.

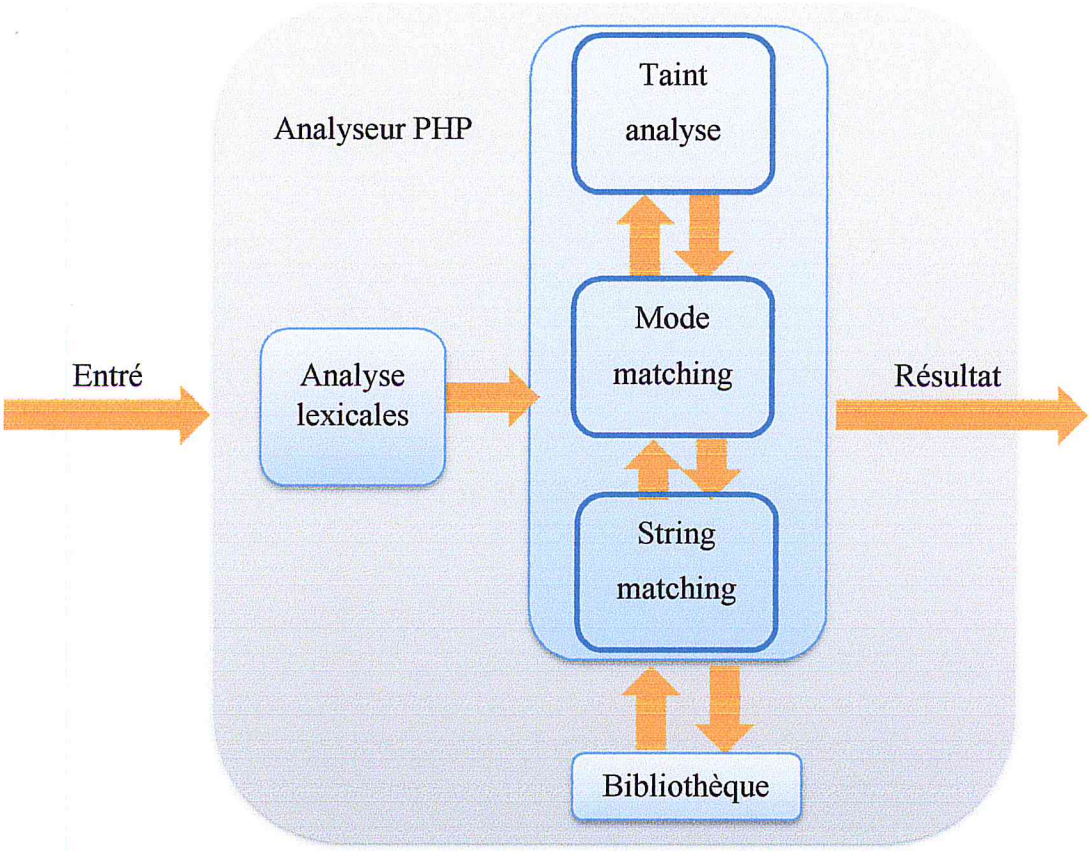


Figure 14: structure de l'analyseur PHP proposé.

On voit que l'analyseur proposé est divisé en deux phases et que chaque phase inclue un ensemble de méthode (figure 15), la première réalise une analyse lexicale, ensuite, les résultats obtenus de cette analyse sont envoyés à la seconde phase, dans cette dernière on voit que les trois méthodes (taint analyse, model matching et string matching) sont utilisées simultanément pour analyser le code source et détecter les failles de sécurité, ces trois techniques se basent sur une bibliothèque qui contient tous les mots clés comme les identificateurs des variables et les noms des fonctions dangereuses et obsolètes..., une fois le résultat obtenu l'analyseur envoi ces résultats à la couche scanner.

A. L'entrée de l'analyseur :

Comme expliqué plus haut nous savons que la couche noyau peut contenir plusieurs analyseurs et que chaque analyseur peut être conçu d'une manière différente pour analyser des langages différents, et que l'ajout d'un autre analyseur par la suite doit être simple et facile, pour cela nous avons proposé d'unifier la structure d'entrée de l'analyseur, ce qui veut dire que pour n'importe quel analyseur la structure de l'entrée attendue est la même, et que tous les analyseurs développés par la suite doivent prendre en considération cette restriction.

- **Description de la structure d'entrée :**

  Dans la description de la couche scanner, on a dit que cette dernière va générer une liste contenant le chemin où se trouve les fichiers du répertoire qui doivent être analysés, par la suite, la couche scanner va envoyer à l'analyseur le chemin d'un seul fichier à la fois et non pas tous les chemins générés, ce qui veut dire que l'entrée de l'analyseur correspond à une chaîne de caractère qui représente l'endroit où se trouve le ficher à scanner.

Ensuite, l'analyseur renvoi un résultat du scanne à la couche deux, et de façon récursive cette même couche va envoyer le chemin d'un autre fichier qui n'est pas encore scanné, et ainsi de suite jusqu'à ce que tous les fichiers soient traités (figure 16).

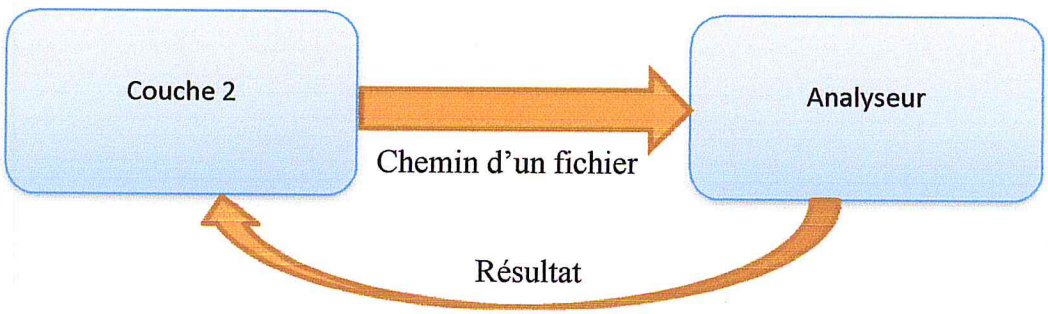


Figure 15: Entrée de l'analyseur.

B. Techniques utilisées :

Nous avons utilisé quatre techniques pour réaliser notre analyseur PHP, et pour mieux comprendre l'utilité de ces techniques nous allons étudier une portion de code PHP (figure 17) comme étude de cas.

```php
<?php
    include("un_autre_fichier.php");
    // c'est un commentaire
    function test($x , $z){
        $z = 6;
        return $x.$z;
    }
    $a = $_GET["id"];
    $z = test($a , $b);
    $c = $_GET['nom']; // $c est infectée
    $d = $c  // $d devient infectée par $c
    $e = 'propagation'.$c;
    $a = htmlspecialchars($a);
    echo $Z;
    function test2(){
        $a = $_GET['test2'];
    }

    function test3(){
        $a;
    }
    mysql_query("SELECT * FROM exmp WHERE id = '$a' ");

?>
```

Figure 16: Portion de code PHP.

1. Analyse lexicale :

Cette méthode est très utilisée dans le domaine de la compilation, elle permet de décomposer une chaîne de caractères en lexèmes, unités ou entités lexicales, aussi appelées tokens en anglais. Nous l'avons utilisée telle qu'elle est définie et utilisée dans les autres domaines, le tableau 14 ci-dessous représente les unités lexicales des quatre premières lignes de la portion de code de la figure 17.

Unité lexicale	Syntaxe
T_OPEN_TAG	< ?php
T_INCLUDE	Include()
T_CONSTANT_ENCAPSED_STRING	"un_autre_fichier.php"
T_COMMENT	// c'est un commentaire
T_FUNCTION	Function
T_STRING	Test
T_VARIABLE	$x
T_VARIABLE	$z

Tableau 14: Quelques unités lexicales du langage PHP.

- **Remarque :**

  - Dans PHP les séparateurs comme « **;** » et «**(**» ou bien « , » ne sont représentés par aucune unité lexicale

Après avoir réalisé une analyse lexicale du code source, tous les caractères illustrés dans le tableau 15 seront supprimés, car ces unités-là sont inutiles pour la détection de failles de sécurité, alors, les enlever nous permettra de gagner en temps de traitement du code source donc au lieu que l'analyseur perde de précieuses millisecondes à traiter ces caractères nous avons préféré les supprimer pour que l'analyseur traite d'autres unités beaucoup plus importantes, ces unités lexicales seront utilisées dans les autres méthodes.

Unité lexicale	Syntaxe
T_DOC_COMMENT	/** */
T_COMMENT	// ou #, et /* */
T_INLINE_HTML	texte en dehors de PHP
T_WHITESPACE	\t \r\n
T_OPEN_TAG	<?php, <? or <%
T_CLOSE_TAG	?> or %>

Tableau 15: Les unités lexicales inutiles pour notre travail.

2. Taint analyse :

Comme nous l'avons vu dans le premier chapitre le Taint analyse consiste à tracer toutes les variables infectées par une entrée utilisateur, par exemple si dans une page d'un site donné il y'a un formulaire ou il existe un champ « nom », une fois que le champ est rempli par un utilisateur et envoyé au serveur, ce dernier va stocker l'information (contenue dans « nom ») dans une variable « $c », cette dernière va être infectée (tagué ou bien tainté) et doit être tracé dans tout le code source (figure 18). Dans le langage PHP on peut accéder aux entrées utilisateurs par des variables super globales, voici ci-dessous une liste de toutes ces variables, les données récupérées depuis ces variables sont considérées comme des données non fiables :

- $_SERVER
- $_GET
- $_POST
- $_REQUEST
- $_FILES
- $_COOKIE

```
10      $c = $_GET['nom']; // $c est infectée
```

Figure 17: Exemple d'une variable infectée.

Notre analyseur ne se base pas que sur la détection des points d'accès, il détecte aussi toutes les variables qui ont pu être contrôlé par l'utilisateur et savoir comment ces variables sont utilisées dans le code source, comme le montre l'exemple de la figure 19 ci-dessous, la variable « $d» qui n'est pas infectée reçoit par une opération d'affectation la variable « $c » qui elle est marquée comme infecter, alors la variable « $d » sera aussi marquée comme étant une variable infectée, cela s'appelle la *propagation d'infection* **(Taint propagation).**

```
10      $c = $_GET['nom']; // $c est infectée
11      $d = $c  // $d devient infectée par $c
```

Figure 18: Exemple de propagation de l'infection.

2.1.Règle de propagation de l'infection (Taint propagation) :

Nous avons vu que l'infection peut se propager à travers les variables dans le code source, pour que notre outil détecte toutes les variables qui ont été infectées par l'utilisateur nous avons défini l'ensemble des règles suivantes :

2.1.1. La concaténation :

Comme montré dans l'exemple ci-dessous (figure 20), la variable « $e » reçoit la chaîne de caractère « 'propagation' » qui est concaténée avec une variable déjà marquée « $c », par conséquent la variable « $e » est automatiquement considérée comme infectée.

```
12      $e = 'propagation'.$c;// $c est déjà infectée
```

Figure 19: Exemple de concaténation.

2.1.2. Les variables *:*

Notre analyseur doit pouvoir faire la différence entre une variable locale et une variable globale, comme illustrée dans l'exemple ci-dessous (figure 18) $a qui se trouve dans la fonction test2 est infectée car elle reçoit une variable GET, cependant la variable « $a » qui se trouver dans la fonction test3 n'est pas infectée car les deux variables ont le même nom mais ne sont pas identiques (deux fonctions différentes), par conséquent la variable « $b » ne sera pas infectée.

```
15      function test2(){
16          $a = $_GET['test2']; //$a est infectée
17      }
18      function test3(){
19          $b = $a; // $b n'est pas infectée
20      }
```

Figure 20: Exemple variable global et local.

2.1.3. Les paramètres des fonctions :

Notre outil doit aussi pouvoir tracer l'utilisation des paramètres dans une fonction et voir si les paramètres ont été infectés ou non comme le montre l'exemple qui suit (figure 22)

```
24  function test($x, $z){
25      mysql_query("SELECT * FROM exmp WHERE id= '$x'");
26      echo $z;
27      return $x.$z;
28  }
29  $a = $_GET["id"];//$a est infectée
30  $b = "Bonjour";// $b n'est pas infectée
31  $z = test($a,$b);// le parametre $x est infectée
32                   // le parametre $z n'est pas infectée
```

Figure 21: Exemple d'infection des paramètres de fonction.

2.1.5. Les résultats des fonctions :

Comme les paramètres des fonctions, les résultats des fonctions peuvent être infectés par les données des utilisateurs donc il faut tracer aussi toutes les fonctions qui ont un résultat infecté comme montré dans l'exemple qui suit (figure 23).

```
24  function test($x, $z){
25      mysql_query("SELECT * FROM exmp WHERE id= '$x'");
26      echo $z;
27      return $x.$z;//le resultat est infecté
28  }
29  $a = $_GET["id"];
30  $b = "Bonjour";
31  $z = test($a,$b);// le parametre $z   est    infectée
```

Figure 22: Exemple des résultats infectés.

2.2. Les règles de remédiation :

Nous avons vu comment les infections peuvent se propager à travers du code PHP, et comme dans beaucoup d'autres langages, il existe un ensemble de fonctions pouvant valider les entrées des utilisateurs et rendre leur utilisation fiable, par exemple la fonction « htmlspecialchar () », mais dans la plupart des cas, ces fonctions peuvent valider les entrées utilisateurs contre un seul type de vulnérabilité, la fonction « htmlspecialchar () » est utilisée pour valider les données contre une attaque XSS seulement (figure 24).

La solution qu'on propose à travers cet outil est de construire pour chaque variable une liste des différentes menaces (failles), afin d'offrir à la variable une sécurisation optimale contre toutes les vulnérabilités répertoriées, par exemple si cette liste est vide la variable peut contenir tout type d'attaques, mais si elle contient une faille « XSS» alors la variable ne peut pas contenir une vulnérabilité «XSS» cependant, elle n'est pas sécurisée contre les autres types d'attaques.

```
12      $a = htmlspecialchars($a);// $a est protégé contre une attaque xss
```

Figure 23: portion de code de remédiation.

3. Model Matching (concordance des modèles) :

On a eu recours à cette méthode «Model Matching » pour pouvoir détecter les vulnérabilités, les variables infectées par les données des utilisateurs (Taint analyse) et l'inclusion de fichier ...etc. Dans ce qui suit, nous allons présenter les trois principaux modèles qu'on a utilisés :

3.1. Modèle pour détecter l'infection des variables par les données d'utilisateur :

On utilise le modèle suivant pour pouvoir dire si une variable est infectée ou non, ce modèle passe par trois principales étapes figure 25.

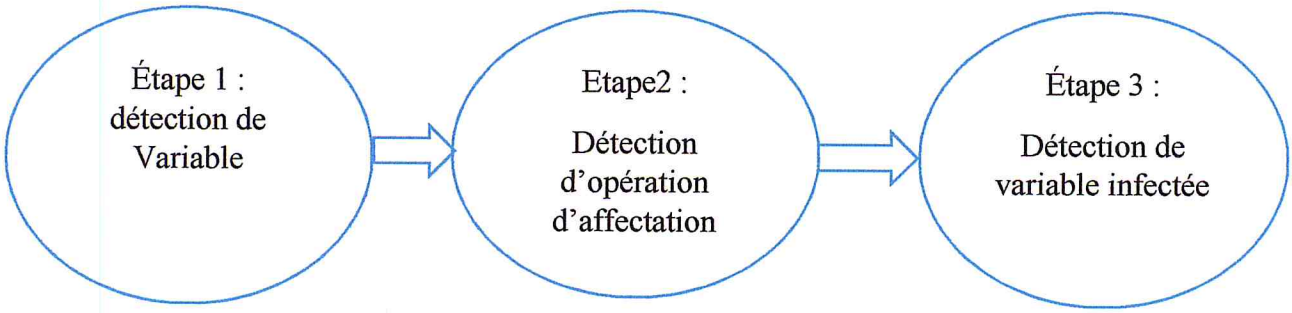


Figure 24: modèle pour la détection des variables infectées.

Comme le montre la figure ci-dessus, pour détecter une variable infecter notre analyseur doit passer par les trois étapes suivantes :

- *Étape 1* : détecter en premier lieu une variable (la variable détectée peut être infectée ou pas).
- *Étape 2* : l'analyseur doit trouver un des opérateurs d'affectation comme « = » ou bien « := ».
- *Étape 3* : notre analyseur doit pouvoir trouver une autre variable, mais cette dernière doit être soit une des variables super globales utilisées pour accéder aux entrées utilisateurs, soit une variable qui est déjà infectée, ou bien il doit trouver un appel à une fonction qui a été marqué comme une fonction avec un résultat infecté (figure 23).

Si notre analyseur passe par ces trois étapes alors il va considérer que la variable détectée dans la première étape est infectée (figure 26).

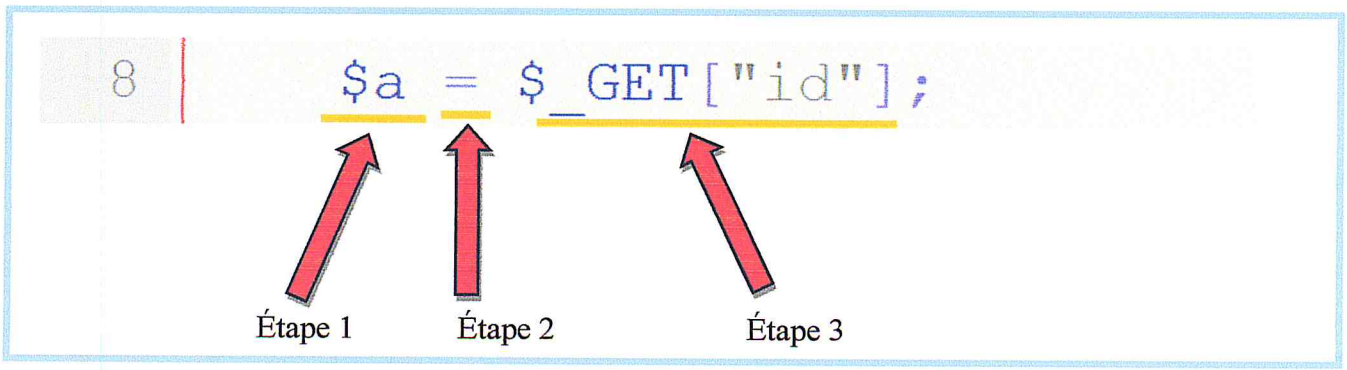


Figure 25: exemple du modèle pour détecter l'infection.

3.2. Modèle pour détecter les vulnérabilités :

Dans ce modèle on a défini comment notre analyseur peut reconnaitre une vulnérabilité, ce modèle est composé de deux étapes seulement figure 27.

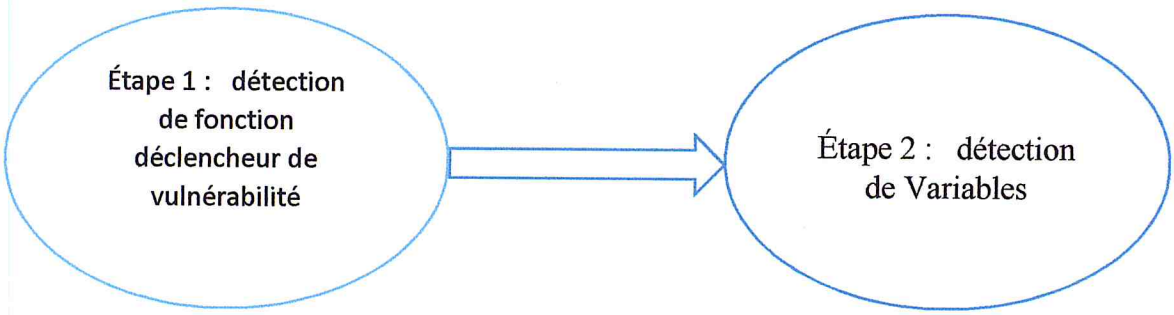


Figure 26: modèle de détection de vulnérabilités.

- Dans la première étape l'analyseur doit détecter un appel à une fonction déclencheur de vulnérabilités, comme son nom l'indique une fonction déclencheur de vulnérabilités c'est une fonction qui va permettre de réaliser une attaque par exemple pour réaliser une attaque XSS le programme doit faire un appel à une fonction qui va retourner au navigateur du code HTML comme « echo », « exit » ou bien « print ».
- Après cette étape notre outil doit trouver une variable qui est infectée pour pouvoir détecter une vulnérabilité.

Dans le code source qu'on a proposé pour l'étude de cas (figure 17) on trouve une faille de type SQL INJECTION dans la ligne 22 (figure 28)

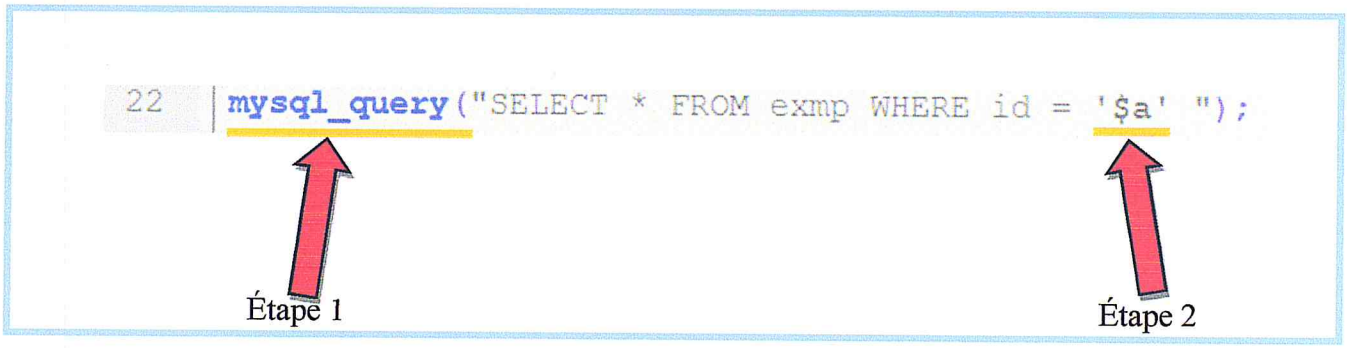


Figure 27: Exemple du modèle pour détecter une vulnérabilité SQL INJECTION.

3.3. Modèle pour détecter les inclusions de fichier :

Dans tous les langages de programmation, le programmeur doit pouvoir inclure des fichiers qui contiennent du code, ces nouveaux fichiers qu'il a inclus peuvent aussi contenir des

vulnérabilités ou des variables infecter d'où la nécessité que notre analyseur doit pouvoir charger et scanner ces fichiers. Pour cela on a défini le mode suivant :

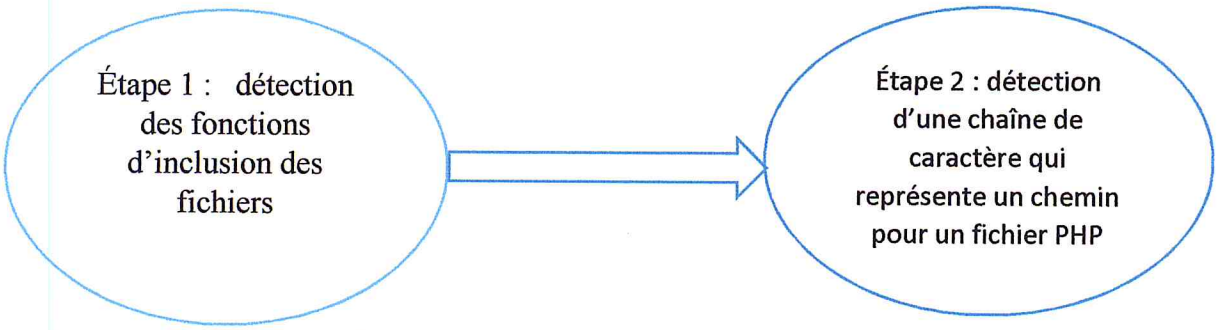


Figure 28 : mode détection de vulnérabilité.

Ce modèle se compose des deux étapes suivantes :

- L'analyseur doit détecter en premier lieu une des fonctions PHP qui permettent l'inclusion d'un fichier, voici ci-dessous une liste de ces fonctions :
  - include
  - include_once
  - require
  - require_once
- En deuxième lieu il doit détecter une chaîne de caractère qui représente un chemin pour un fichier PHP, ce qui veut dire une chaîne de caractère qui se termine par « .php ».

Remarque : après que notre analyseur passe par ces deux étapes, il doit vérifier que le fichier à inclure existe pour inclure et scanner ce nouveau fichier.

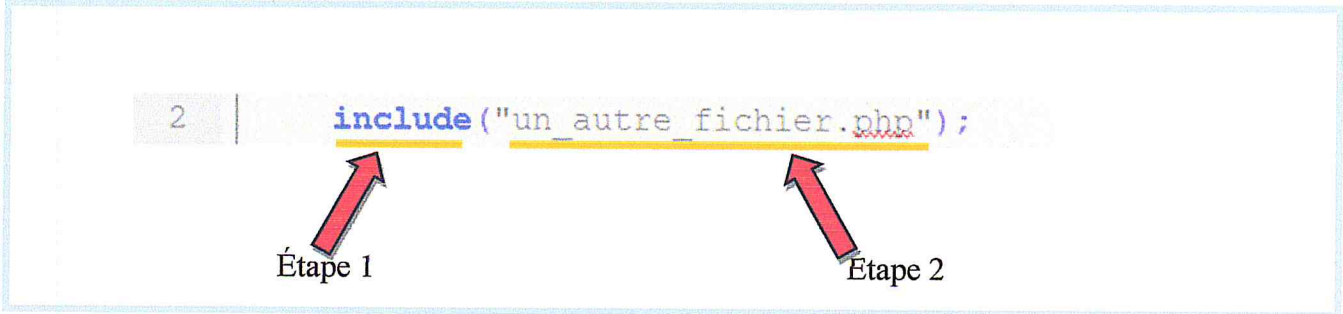


Figure 29: exemple du modèle pour une inclusion d'un fichier.

4. String Matching (concordance de chaînes de caractères) :

Le string matching est utilisé dans les deux techniques précédentes (taint analyses, models matching), pour pouvoir identifier les mots clés comme les noms des variables super globales pour accéder aux données des utilisateurs et les mots clés pour identifier un modèle de vulnérabilité, etc.

Nous utilisons aussi le string matching pour détecter l'utilisation de toutes les fonctions dangereuses ou obsolètes qui concernent le langage PHP, nous noterons aussi que même si les fonctions obsolètes ont été supprimées de PHP certain projet peuvent utiliser une ancienne version qui prend toujours en compte ces fonctions-là c'est pour cela que notre analyseur les prend en charge, voici quelques-unes parmi plusieurs que nous avons répertoriés dans la bibliothèque de notre analyseur (tableau 16 et 17).

Fonction dangereuse	**Description**
apache_child_terminate	Cette fonction programme le processus Apache pour qu'il se termine une fois la requête PHP courante terminée.
apache_setenv	définit la valeur de la variable d'environnement Apache spécifiée par le paramètre variable.
define_syslog_variables	elle initialise toutes les variables relatives aux fonctions syslog.
Exec()	Cette fonction exécute un programme externe.
Eval()	La fonction eval exécute une chaîne comme un script PHP, son utilisation est vivement déconseillée.
shell_exec()	Exécute une commande via le Shell et retourne le résultat sous forme de chaîne

Tableau 16: liste de quelques fonctions dangereuses dans PHP.

Fonction obsolète	**Description**
call_user_method	Appelle une méthode utilisateur sur un objet donné, cette fonction est devenue obsolète en PHP 4.1.0, et a été supprimée en PHP 7.0.0
Dl	Charge l'extension PHP **library** à la volée, elle a été supprimée du SAPI en PHP 5.3.
Ereg	Elle fait une recherche par expression rationnelle standard, cette fonction est devenue *obsolète* en PHP 5.3.0, et a été *supprimée* en PHP 7.0.0.
eregi_replace	Cette fonction effectue une recherche par expression rationnelle, cette fonction est devenue *obsolète* en PHP 5.3.0, et a été supprimée en PHP 7.0.0.

Split	Scinde une chaîne en un tableau, grâce à une expression rationnelle, cette fonction est devenue *obsolète* en PHP 5.3.0, et a été supprimée en PHP 7.0.0.
mysql_db_query	Sélectionne une base de données et y exécute une requête, cette fonction était obsolète en PHP 5.3.0, et la totalité de l'extension originale MySQL a été supprimée en PHP 7.0.0. À la place

Tableau 17: liste de quelque fonction obsolète dans PHP.

C. La bibliothèque :

Nous avons défini une bibliothèque pour enregistrer tous les mots clés et les noms de fonctions, dont l'analyseur à besoin (Tableau 14,15 et 16), en plus de cela, elle est alimentée par une liste qui contient la définition de chaque vulnérabilité. Une vulnérabilité est définie par les points suivants :

- Le nom de la vulnérabilité (exemple :XSS).
- Une liste des noms de fonction qui permet de déclencher cette vulnérabilité (echo, exit …etc.).
- Une liste des noms des fonctions qui permet de remédier à cette vulnérabilité (htmlspecialchars, htmlentities…etc.).
- Une description de cette vulnérabilité.
- Une remédiation à cette vulnérabilité.

L'analyseur va se baser sur cette liste pour identifier les vulnérabilités, cela va faciliter l'ajout d'autres vulnérabilités de type inclusion par la suite, car l'ajout ne se fera qu'en fonction de cette liste au niveau de la bibliothèque.

D. Structure des résultats d'un analyseur :

Comme mentionné plus haut le résultat de l'analyseur doit être le même pour tous les analyseurs afin faciliter la gestion des résultats dans la coucher scanner, pour cela on a défini la structure des résultats suivante :

- Un seul résultat doit être envoyé à la couche 2 (couche scanner).
- Chaque vulnérabilité doit contenir les informations suivantes :
  - Nom de la variable qui a causé la vulnérabilité.
  - La ligne de code ou la vulnérabilité a été détectée.
  - Le fichier ou la vulnérabilité a été détecté.
  - Une description sur l'utilisation de la variable qui a causé la vulnérabilité.

9 Conclusion :

À l'issue de ce chapitre, nous avons en premier lieu exposé ce que le système existant comportait comme fonctionnalités tout en spécifiant ses objectifs, ses points forts et ses points faibles, mais aussi nous avons décrit l'interaction que devra avoir le système actuel avec celui qu'on a proposé et cela en décrivant la base de données qui en l'occurrence représente le cœur même de cette interaction, nous avons ensuite suivi une solution fondée sur l'analyse et la conception d'un système de couches et de module afin de spécifier et de décrire le fonctionnement de l'analyseur statique du code source, chaque modules est décrit par les fonctionnalités et les techniques qu'il propose et cela selon la couche dans lequel il se trouve, nous avons aussi énuméré et définis toutes les techniques que nous avons utilisées afin de mettre en place notre analyseur d'audit code source tout en spécifiant la bibliothèque qui comporte la liste des vulnérabilités et des fonctions traitées par notre système d'audit. De plus, le système permet à l'expert de mettre à jour les vulnérabilités détectées ainsi que leur suppression, pour finir nous pouvons dire que nous avons cerné les objectifs de notre système. Pour les atteindre, nous avons suivi une solution fondée sur l'analyse et la conception d'un système de couches et de module combinés a plusieurs techniques travaillant ensemble dans le but d'assurer l'audit en général.

Dans le chapitre qui suit, nous allons présenter l'environnement de travail et les outils qu'on va utiliser pour implémenter le système étudié, nous allons aussi décrire l'application conçue d'une manière générale, tout en évoquant ses différentes interfaces.

Chapitre IV : Test et Validation

1 Introduction

Dans ce chapitre, nous allons présenter l'environnement de développement de notre système en précisant les outils et les langages de programmation qu'on a choisie pour le réaliser. Juste après, on va présenter l'implémentation des solutions et les fonctionnalités proposées dans le chapitre précédent, ainsi que les principales interfaces qui composent le système actuel et les celles du système proposé.

2 Environnement de travail :

Nous avons utilisé les outils suivant afin de mettre au point notre système proposé :

2.1 Les outils :

Afin de réaliser notre système, nous avons utilisé plusieurs outils et langages qui nous ont permis de mettre au point une application dynamique et interactive.

Wamp :WampServer est une plate-forme de développement web sous Windows pour des applications web dynamiques (33). Qui comporte :

Le serveur web ***Apache version 2.4.9*** : est un serveur HTTP, il permet d'utiliser des fichiers PHP et d'installer des bases de données en local. Il est stable et performant.

Le serveur de base de données ***MYSQL version 5.6.17*** : est un système de gestion de bases de données relationnelles (SGBDR) robuste et rapide. Il contrôle l'accès aux données pour s'assurer que plusieurs utilisateurs peuvent se servir simultanément d'une même base de données pour y accéder rapidement. (34)

L'interface d'administration ***phpMyAdmin version 4.1.14*** : est un outil logiciel libre écrit en PHP , destiné à gérer l'administration de MySQL sur le web. (35)

Sublime TEXT 3 **:** Est un éditeur de texte conçu principalement pour l'édition d'extraits de code, il se démarque par la quantité et la qualité de ses fonctionnalités. On retrouve notamment la sélection multiple et les curseurs multiples. Ces fonctionnalités et beaucoup d'autres rendront l'édition du code source beaucoup plus rapide et facile. (36)

2.2 Les langages de développement de l'application :

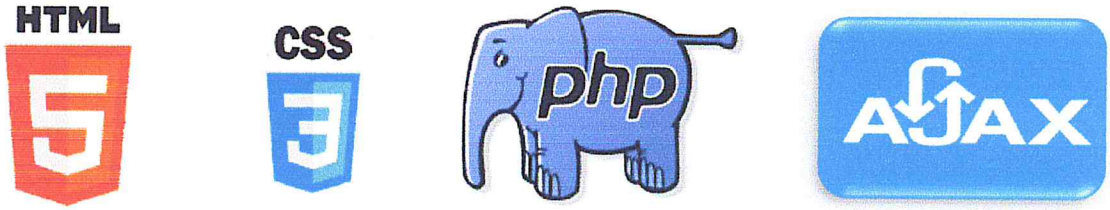


Figure 30 : Les principaux outils de développement du système proposé.

Afin de réaliser notre système, nous avons utilisé plusieurs langages qui nous ont permis de mettre au point un système dynamique et interactif, ces langages sont :

PHP 5 (Hypertext Preprocessor) **:** Est un langage de programmation, très proche syntaxiquement du langage C, destiné à être intégré dans des pages HTML. Contrairement à d'autres langages, PHP est principalement dédié à la production de pages HTML générées dynamiquement. (34)

AJAX (Asynchronous JavaScript and XML) : Derrière ce nom se cache un ensemble de technologies destinées à réaliser de rapides mises à jour du contenu d'une page web, sans qu'elles nécessitent le moindre rechargement visible par l'utilisateur de la page web. Les technologies employées sont diverses et dépendent du type de requêtes que l'on souhaite utiliser, mais d'une manière générale le JavaScript est constamment présent, d'autres langages sont bien entendu pris en compte comme le HTML et le CSS.(sourceopenclassroom). (38)

HTML 5 (HyperText Mark-up Language) **:** C'est un langage de balisage moderne qui corresponde mieux au type d'informations publiées sur le web.Il contient également un ensemble de nouveaux éléments de formulaire qui devraient faciliter considérablement le développement d'applications web. (39)

CSS (Cascading Style Sheets) : Est un langage pour décrire la présentation des pages web,il est indépendant de HTML et peut être utilisé avec n'importe quel langage de balisage XML. La séparation du code HTML CSS rend plus facile à maintenir les sites. (40)

JavaScript : C'est un langage de script par excellence des navigateurs web, il offre la possibilité d'implémenter des traitements élaborés dans des pages web. Il peut être mis en œuvre dans toute application disposant d'un interpréteur pour ce langage. (41)

Bootstrap : Bootstrap est un framework de type "Front-End Framework", c'est une compilation de plusieurs éléments et fonctions web-design personnalisables, ces éléments sont une combinaison de HTML, CSS et JavaScript.

2.3 Choix du Framework :

Un Framework n'est pas indispensable pour la création de notre système web. Cependant, pour que l'application soit robuste, facile à faire évoluer et réalisable en un temps minimum, un framework représente un outil idéal.

Il existe une grande quantité de frameworks PHP. Chacun présentant des avantages et des inconvénients, il a fallu réduire le choix de ces frameworks (figure 32) pour permettre ainsi de ne pas perdre trop de temps sur le comparatif et passer plus rapidement à la phase de modélisation. (42)

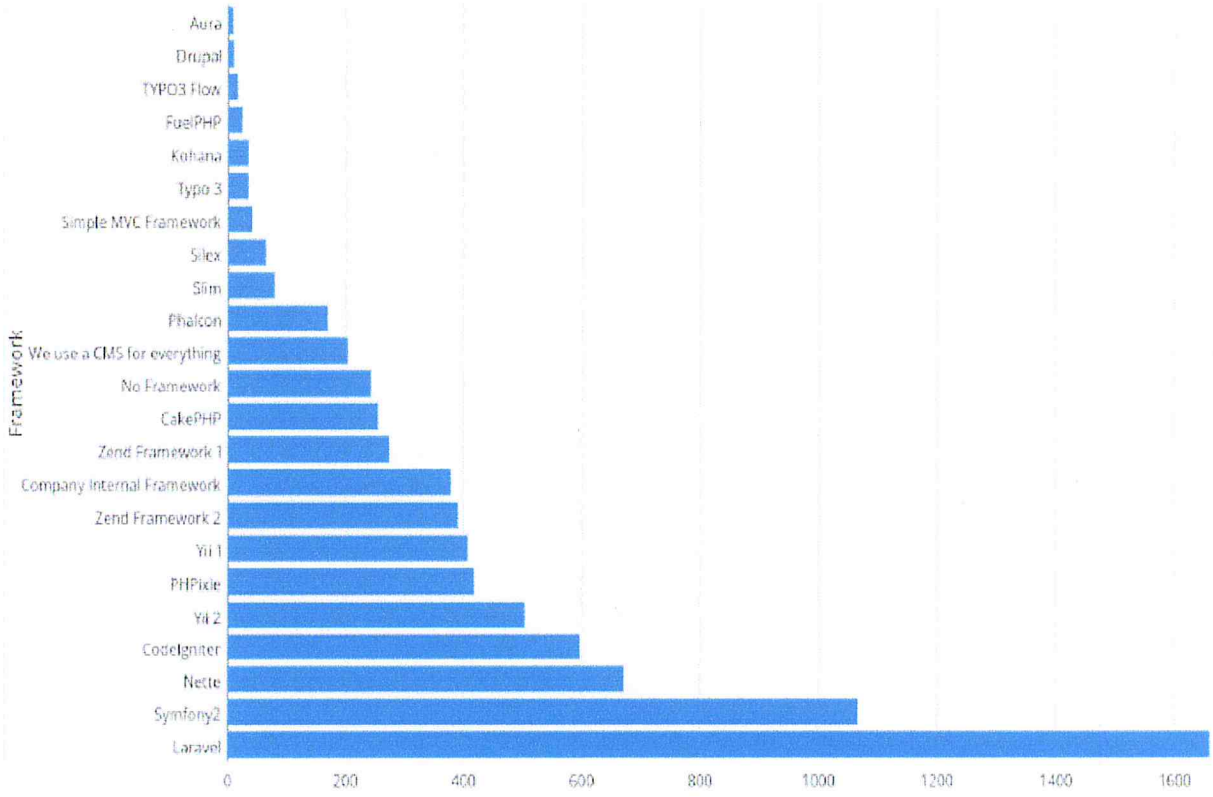


Figure 31 : Popularité des frameworks PHP en entreprise. (43)

La figure ci-dessus présente les frameworks les plus populaires en entreprise. Nous allons donc ici nous concentrer sur le framework le plus populaire et qui est Laravel.

Laravel est un Framework web open source écrit en PHP. Il respecte le modèle MVC (modèle-vue-contrôleur). C'est un Framework orienté objet distribué sous licence MIT.

Laravel est un outil qui, dans sa conception, se base sur le meilleur de plusieurs autres

Frameworks pour développer son propre système et être plus efficace. Il possède également des composants qui lui sont propres.

Il fournit entre autres :

- Un système de routage des vues.
- Un créateur de requêtes SQL et une gestion de version de Base de données.
- Un système d'authentification pour les connexions.
- Un système de cache.
- Une gestion de sessions.

2.4 Le paradigme MVC (Modèle-Vue-Contrôleur) :

Le paradigme MVC est un schéma de programmation qui propose de séparer une application en trois parties :

- ❖ Le modèle, qui contient la logique et l'état de l'application.
- ❖ La vue, qui représente l'interface utilisateur.
- ❖ Le contrôleur, qui gère la synchronisation entre la vue et le modèle.

Le point essentiel consiste à séparer les objets graphiques des objets métier, afin de pouvoir les faire évoluer indépendamment et les réutiliser.

Au final, une telle séparation favorise le développement et la maintenance des applications :

- ❖ Le modèle étant séparé des autres composants, il est développé indépendamment. Le développeur du modèle se concentre sur le fonctionnel et le transactionnel de son application.
- ❖ Le modèle n'est pas lié à une interface, il peut donc être réutilisé. (43)

Remarque : Le framwork Laravel adopte le patron MVC, mais ne l'impose pas.

2.5 Aspect sécurité :

Laravel est l'un des Frameworks les plus sécurisés et le plus corrigés quand il s'agit de failles et de brèche détectés dans le noyau de son implémentation, il propose un très bon système de sécurité qui est généralement déjà implémenté, il suffit alors de le paramétrer correctement pour avoir une application sécurisée à l'épreuve des intrusions.

2.6 Déploiement et contraintes techniques de notre système :

L'application devra être déployée sur un serveur propre à l'entreprise, de plus, la conception et implémentation du système d'audit code source devra être un système multiutilisateur pour permettre à chaque acteur du département de sécurité des systèmes d'information d'effectuer ses tâches. Pour ce faire, nous avons choisi d'utiliser une solution web qui sera régie

par l'architecture MVC (voir plus haut) qui apporte de réels avantages à notre application tels que :

- ✓ Une conception claire et efficace grâce à la séparation des données de la vue et du contrôleur.
- ✓ Un gain de temps de maintenance et d'évolution du système.
- ✓ Une plus grande souplesse pour organiser le développement de l'application entre les différents développeurs.

3 Implémentation du système proposé :

Nous allons présenter dans cette partie du chapitre la façon avec laquelle nous avons implémenté notre solution et cela en décrivant l'implémentation du système et des couches qui le composent.

3.1 Implémentation de la couche utilisateur :

La couche utilisateur qui comme nous l'avons dit dans le chapitre précédant se compose de plusieurs modules, ces modules sont responsables des interactions homme-machine, dans la plupart de ces derniers nous pouvons trouver des fonctionnalités comme :

- Contrôler les données des utilisateurs (Contrôleur).
- Gérer les affichages des données.
- Gestion d'authentification (ce module est prés définit dans le framwork laravel).

L'implémentation des modules de la coucher utilisateur est simple et basic, par conséquent, nous n'allons pas la détailler.

3.2 Implémentation de la couche scanner

Nous avons dit précédemment que la couche scanner est celle qui pilote la couche noyau, nous avons aussi listé dans le chapitre précédant toutes les fonctionnalités de cette couche ainsi que les données qu'elle traite lors de son exécution, ci-dessous on va présenter un pseudo-code (figure 33) pour cette couche :

```
Début : $liste = générer_la_liste_de_fichier($chemin , $langage) ;
        Selon ($langage){  Cas PHP : {
                Pour chaque élement de $liste{
                $res = Annalyseur_php.analyse($liste[$i])}
                } Cas JAVA : {
                Pour chaque élement de $liste{// configuration de
        l'analyseur java si besoin
                $res = Annalyseur_java.analyse($liste[$i]) ;
                }
                }
                ....
                }
                Enregister_dans_la_base_de_donnée($res) ;
FIN
```

Figure 32: Pseudo code illustrant le fonctionnement de la couche scanner.

3.3 Implémentation d'un prototype d'analyseur PHP :

L'analyseur proposé utilise les quatre méthodes qu'on a présentées dans le chapitre précèdent (analyse lexical, taint analyse, model matching et string matching), et se base sur une bibliothèque, cette dernière permet de stocké les informations nécessaires pour réaliser l'analyse du code (mots clés du langage et nom des fonctions … etc.), dans ce qui va suivre nous allons voir comment on a pu implémenter un prototype de l'analyseur pour le langage PHP (cette partie ne contient pas tous les détaille de l'implémentation mais juste les plus importants).

3.3.1 Analyse lexicale :

Après quelques recherches on a pu trouver une fonction définit dans le langage PHP qui permet de réaliser une analyse lexicale d'un code PHP, cette fonction s'appelle « token_get_all () » figure34, nous avons utilisé cette fonction pour réaliser l'analyse lexicale.

```
array token_get_all ( string $source [, int $flags = 0 ] )
```

Figure 33: la fonction "token_get_all."

Cette fonction prend en entrées une chaîne de caractère qui correspond à un code source et retourne en résultat un tableau qui contient les trois informations suivantes :

- L'unité lexicale, c'est une valeur entière positive qui détermine le type de l'unité lexical Exemple : le numéro « 320 » correspond à l'unité lexicale « T_VARIABLE »
- La syntaxe, c'est une chaîne de caractère qui correspond à la syntaxe de l'unité lexicale par exemple « $x »
- Le numéro de ligne de code, il détermine quelle ligne cette unité lexicale a été détectée dans le code

La figure 35 représente le résultat des deux premières lignes de la fonction « token_get_all() » après qu'on lui ait passé la ligne de code de la figure (34).

```
array:108 [▼
  0 => array:3 [▼
    0 => 379
    1 => "<?php "
    2 => 1
  ]
  1 => array:3 [▼
    0 => 382
    1 => """
      \r\n
       \t
      """
    2 => 1
  ]
  2 => array:3 [▼
    0 => 262
    1 => "include"
    2 => 2
  ]
  3 => "("
  4 => array:3 [▼
    0 => 323
    1 => ""un_autre_fichier.php""
    2 => 2
  ]
  5 => ")"
  6 => ";"
```

Figure 34 : exemple du résultat de la fonction « token_get_all() »

Après avoir réalisé l'analyse lexicale, on a traité le résultat de cette fonction pour supprimer toutes les unités inutiles pendant la phase de scan « < ?php », la figure suivante représente le résultat après avoirs effacer les unités lexicales inutiles.

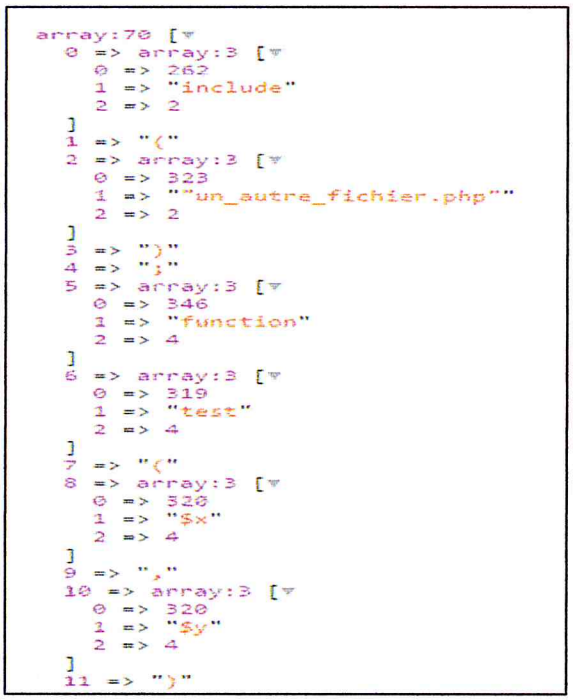


Figure 35 : le tableau des unités lexicales après suppression des unités inutiles

La figure précédente montre le résultat sous forme de tableau et cela après le premier traitement de l'analyse lexicale, chaque élément du tableau contient un autre tableau qui se compose des trois informations concernent l'unité lexicale, on voit aussi que les éléments comme « ; », « (», «) » et d'autre ne sont pas définit de la même manière que les autres unités lexicales mais ils sont défini que par leur syntaxe. On voit aussi sur la figure 36 que le nombre des unités lexicales est plus petit (70 unités lexicales) que le nombre de ces unités dans la figure 35 (108 unités lexicales, cela va permettre un gain de temps important lors de la deuxième phase d'analyse (taint analyse, model matching et string matching).

Après cette étape nous avons proposé de regrouper les éléments lexicaux par ligne, ce qui veut dire que toutes les unités de chaque ligne de code source sera regroupé dans une seule case du tableau (figure37), cela va faciliter l'analyse du code source dans la deuxième phase.

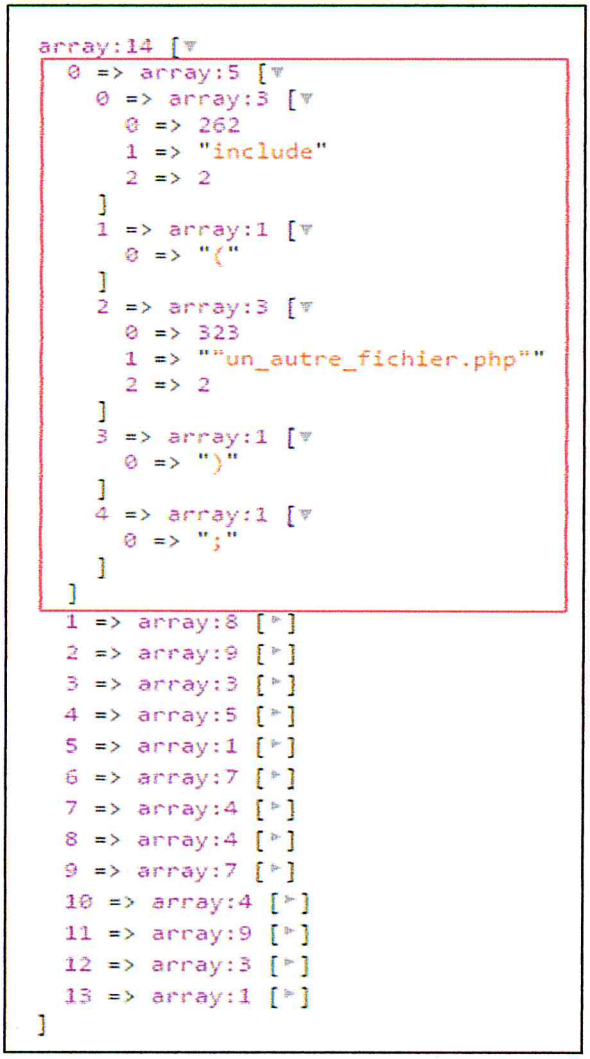


Figure 36: Regroupement des unités lexicales par lignes

3.3.2 **Taint analyse :**

Pour implémenter cette technique on doit parcourir le tableau issu de la première phase à la recherche des variables infectées en utilisant la technique du mode matching (le modèle qui détecte les infections), si on trouve une correspondance de modèle on ajoute la variable infecter dans le tableau des variables infectées, chaque case du tableau des variables infecter contient les informations suivantes :

- Le nom de la variable infecter par exemple $x.
- Le numéro ligne ou la variable a été infecté.
- Le chemin du fichier ou l'infection a été détecté car il y a une possibilité qu'une variable globale soit infectée dans un fichier autre que celui qui est en train d'être scanné
- Une liste des vulnérabilités prouvant que la variable a été sécuriser contre par exemple « $x = htmlspecialchar($x) » la variable $x .ne peut pas contenir une attaque xss.
- La ligne complète du code source par exemple « $x = $_GET['exp'] ; »

La figure ci-dessous (figure 38) est un pseudo-code qui montre l'implémentation du taint analyse dans notre analyseur (ce pseudo-code ne montre pas tous les cas de figure possibles).

```
tokens = analyse_lexicale(); // réalise l'analyse lexicale avec toutes les
modifications
pour ( i =  0 ; i < tokens.length;i++){ // pour chaque ligne de code
      pour ( j = 0 ; j < tokens[i].length ; j++){
        si( token[i][j] est une variable && model1 == 0) {// model1 est le
modèle d'infection
                model1 ++ ;
                info_variable = les_informations_sur_token[i][j] ; //
info_variable contient les informations sur token[i][j] car elle peut être
infectée
        }
        si(token[i][j]  est une opération d'affectation && model1 == 1 ){
                model1++
        }
        si( token[i][j] est une variable && model1 == 2 &&  token[i][j] est
une variable infecter  ) {
                ajouter info_variable dans le tableau des variables infectées
;
        }
      }
}
```

Figure 37 : Portion code source du taint analyse

3.3.3 Model matching (modèle de détection de vulnérabilité) :

Comme on la dit dans le deuxième chapitre, pour détecter une vulnérabilité notre analyseur doit passer par les deux étapes suivantes :

1. Détection d'une fonction déclencheur de vulnérabilité.
2. Détection d'une variable infectée.

Pour implémenté ce modèle on a utilisé une simple variable entière qui est initialisée a zéro et on incrémente cette variable d'un chaque fois que l'analyseur passe par une étape du mode. Si la ligne se termine et la variable est égale à un on réinitialise la variable à zéro, et si cette dernière est égale à deux cela implique qu'une vulnérabilité est détectée.

Lorsqu'on détecte une vulnérabilité on l'ajoute dans un tableau qui contient les vulnérabilités trouver ce tableau est initialement vide, chaque case du tableau contient les informations suivantes :

- Le nom de la vulnérabilité
- La variable infecter qui a causé cette vulnérabilité
- Le numéro de ligne de la vulnérabilité
- Une description sur le déclenchement de la vulnérabilité

3.3.4 **La bibliothèque :**

On a implémenté la bibliothèque comme étant une classe avec des tableaux qui contiennent les informations suivantes (figure 39) :

- Les unités lexicales qui sont inutiles
- Les variables qui permettent l'accès aux données utilisateur
- Les fonctions qui permettent d'inclure des fichiers comme « include() »
- Une liste de toutes les vulnérabilités, cette liste contient deux informations
    - Le nom de la vulnérabilité
    - Une liste des fonctions déclencheurs de cette vulnérabilité
- Une liste des fonctions de remédiation cette liste contient deux informations
    - Le nom de la vulnérabilité
    - Une liste des fonctions de remédiation de cette vulnérabilité

```
public static  $T_IGNORE = array(
        #T_BAD_CHARACTER,
        T_DOC_COMMENT,
        T_COMMENT,
        //T_ML_COMMENT,
        T_INLINE_HTML,
        T_WHITESPACE,
        T_OPEN_TAG,
        T_CLOSE_TAG
    );
    public static $T_INPUT = array(
        '$_GET',
        '$_POST',
        '$_REQUEST',
        '$_FILES',
        '$_COOKIE',
        '$_SERVER',
    );

    public static $T_INCLUDE = array(
        T_INCLUDE,
        T_REQUIRE_ONCE,
        T_INCLUDE_ONCE,
        T_REQUIRE,
        );

    public static $SECURING_FUNCTION = array(
            array(
                    'secure_against'=>'XSS',
                    'function_names'=>array(
                            'htmlspecialchars',
                            'htmlentities',
                        ),
            ),
            array(
                    'secure_against'=>'CMD INJECTION',
                    'function_names'=>array(
                            'escapeshellcmd',
                            'escapeshellarg',
                        ),
            ),
    );
```

Figure 38: une portion de code de l'implémentation de la bibliothèque.

On a vu que l'analyseur se base que sur une liste de définition des vulnérabilités pour pouvoir détecter ces derniers, voici ci-dessous la liste des vulnérabilités que notre analyseur peut détecter pour l'instant :

- REFLECTED XSS
- CODE INJECTION
- SQL INJECTION
- FILE INCLUSION
- DIRECTORY TRAVERSAL
- CMD INJECTION
- STORED XSS

Notre analyseur peut détecter que les sept vulnérabilités citées au-dessus, mais il est très simple d'ajouter d'autres vulnérabilités dans la liste de définition des vulnérabilités(la figure 40, 41). C'est un exemple d'ajout d'une vulnérabilité appelé « vul », cette dernière est déclenchée par les fonctions suivantes « d_vul_1 et d_vul_2 » et pour la remédiation contre cette vulnérabilité il existe deux fonctions « r_vul_1 et r_vul_2 ».

```
public static $VULNERABILITYS_1 = array(
    array(
            'vulnerability_name'=>'REFLECTED XSS',
            'vulnerability_alert'=> array(
                    'echo',
                    'print',
                    'exit',
                ),
    ),
    array(
            'vulnerability_name'=>'SQL INJECTION',
            'vulnerability_alert'=> array(
                    'mysql_query',
                ),
        ),
    array(
            'vulnerability_name'=>'STORED XSS',
            'vulnerability_alert'=> array(
                    'mysql_query',
                ),
        ),
    array(
            'vulnerability_name'=>'CMD INJECTION',
            'vulnerability_alert'=> array(
                    'exec',
                    'passthru',
                    'system',
                    'shell_exec',
                    'proc_open',
                    'popen',
                    'pcntl_exec',
                ),
        ),
    array(
            'vulnerability_name'=>'vul',
            'vulnerability_alert'=>array(
                    'd_vul-1',
                    'd_vul_2',
                ),

        );

);
```

Figure 39: Ajout d'une vulnérabilité (partie 1).

```
public static $SECURING_FUNCTION = array(
        array(
                'secure_against'=>'XSS',
                'function_names'=>array(
                        'htmlspecialchars',
                        'htmlentities',
                    ),
        ),
        array(
                'secure_against'=>'CMD INJECTION',
                'function_names'=>array(
                        'escapeshellcmd',
                        'escapeshellarg',
                    ),
        ),
        array(
        'secure_against'=>'vul',
        'function_names'=>array(
            'r_vul_1',
            'r_vul_2',
            ),
        ),
);
```

Figure 40: Ajout d'une vulnérabilité (partie 2).

Nous avons dit auparavant dans la conception de la solution qu'il y a une seule liste qui contient la définition et les informations concernant les vulnérabilités, cependant l'implémentation en général se confronte à des restrictions de tous genres et pour cette même raison nous avons été dans l'obligation de diviser cette liste en trois sous liste différentes afin de permettre une meilleure structure du code.

4 Test du système proposé :

Pour réaliser le test de notre analyseur, on suppose le scénario suivant :

- Une demande d'audit d'un projet appelée test est envoyée au système actuel d'ELIT.
- Le responsable affect le projet a un auditeur.
- L'auditeur entre dans notre système et trouve qu'une nouvelle demande lui est attribuée (figure 44).
- L'auditeur télécharge depuis le serveur du client le code source de son projet (cette phase se fait manuellement).
- L'auditeur alors va trouver une hiérarchie dans le projet que l'analyseur va parcourir et cela, fichier par fichier.

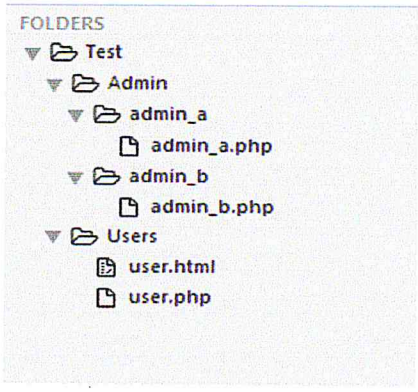


Figure 41: Hiérarchie du projet à auditer.

Nous remarquons que dans la figure ci-dessus (figure 42) le projet « Test » contient quatre fichiers PHP et un fichier HTML (*user.html*).

Remarque : Le code source de ces fichiers ont été proposées par ELIT pour réaliser les tests.

Dans cette partie nous allons voir les interfaces qui constituent notre application tout en détaillant chaque fonctionnalité et le la façon avec laquelle notre système procède.

Premièrement, nous allons commencer par l'affectation du projet aux auditeurs qui découlent de la demande d'audit, comme le montre l'interface ci-dessous :

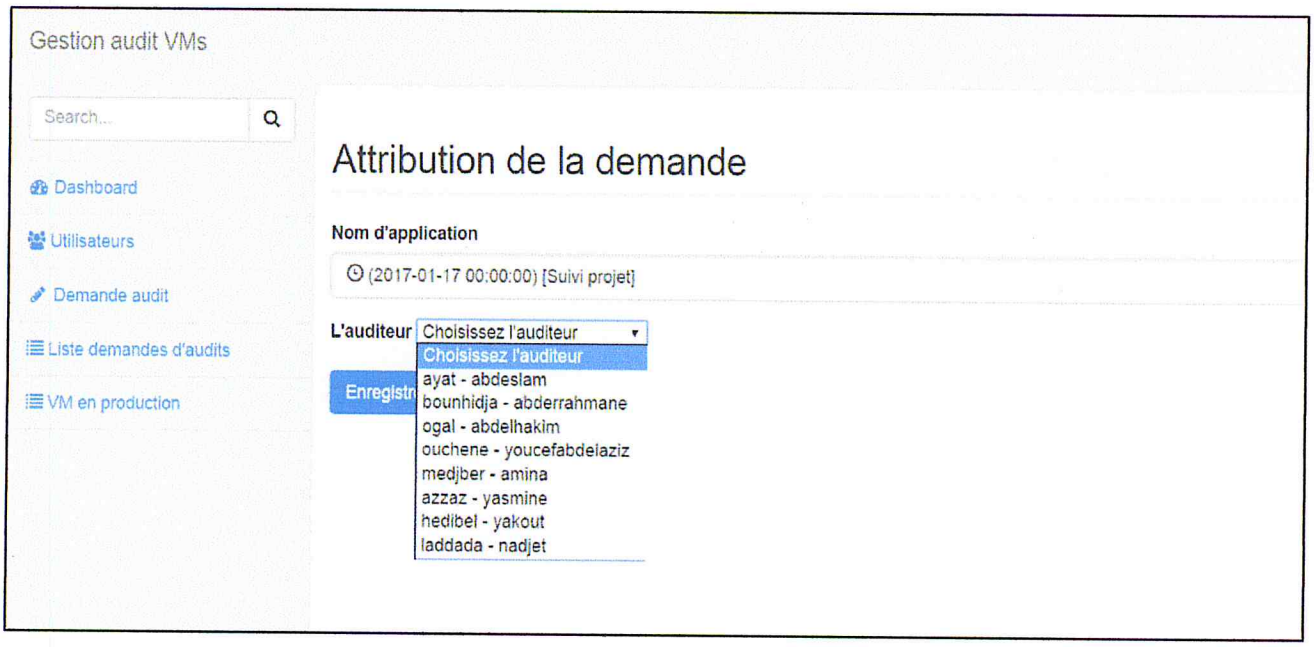


Figure 42: Attribution de projet à auditer aux auditeurs.

Remarque : l'interface illustrée dans la figure 43 n'appartient en aucun cas à notre système, cette interface appartient au système déjà existant, quand la demande d'audit est attribuée à un ou plusieurs auditeurs, notre système d'audit entre en jeu.

Après que l'auditeur se soit connecté il sera automatiquement dirigé vers la page ou s'afficheront tous les projets qui lui ont été affectés précédemment par le système actuel, alors l'auditeur n'aura plus qu'à sélectionner le projet qu'il voudra analyser dans notre cas nous avons un seul projet (figure 44)

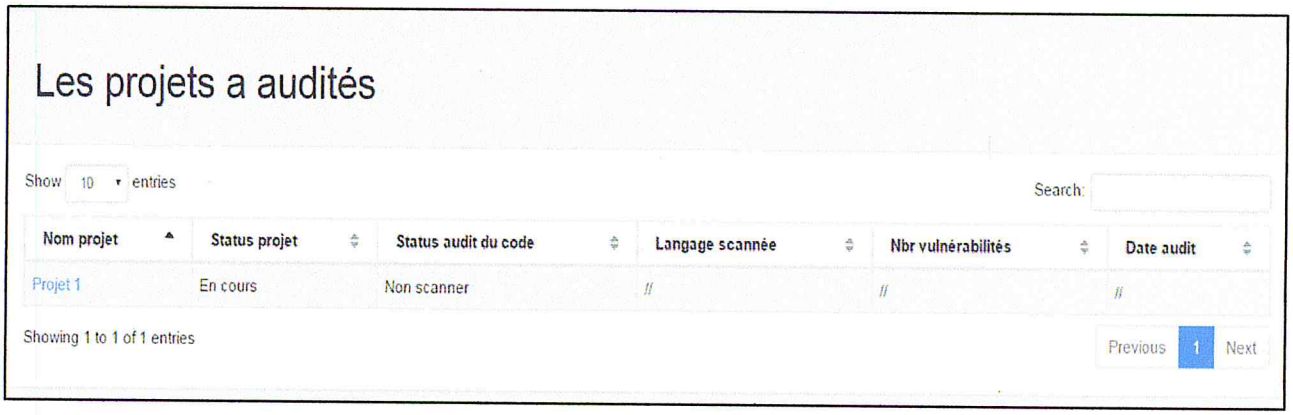


Figure 43: Interface des projets affectés à l'auditeur.

Lorsque l'auditeur sélectionnera le projet qu'il voudra analyser et générer un rapport l'interface qui suit s'affichera

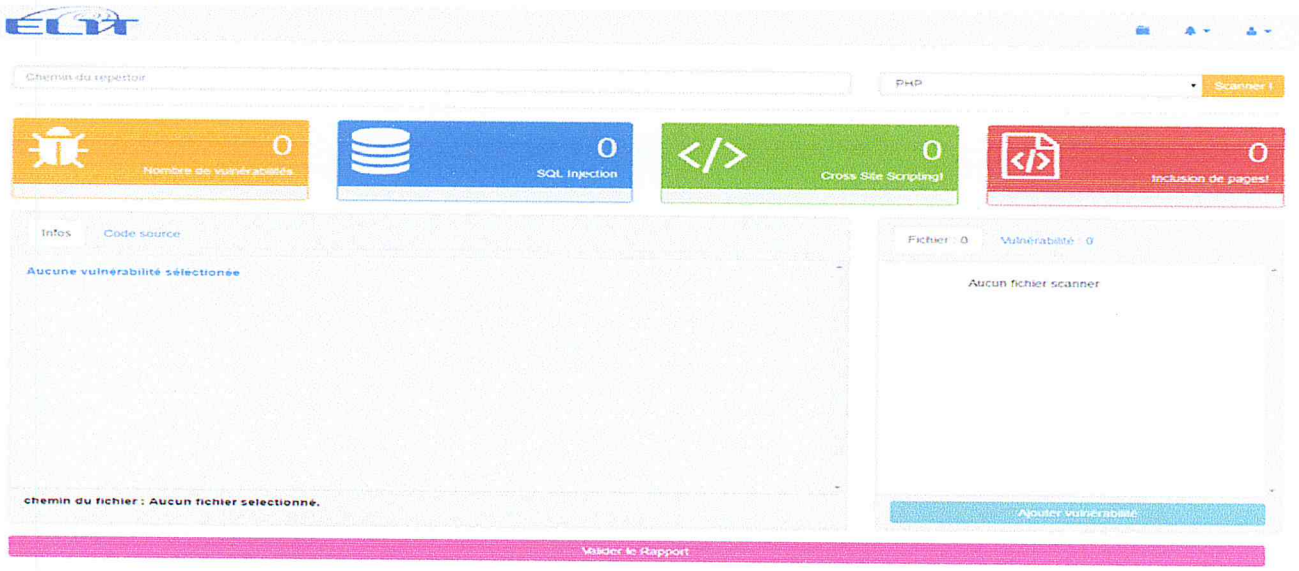


figure 44 : interface de scan.

Alors l'auditeur aura la possibilité de sélectionner le langage de programmation que l'analyseur devra scanner ainsi que le chemin du répertoire dans lequel le projet se trouve.

Nous verrons par la suite le rôle de chaque partie ainsi que les interfaces du résultat et l'implémentation de l'analyseur PHP.

- **Résultat attendu :**

Après le scan, notre analyseur doit détecter trois fichiers PHP et ignorer le fichier HTML, dans chacun des fichiers PHP il doit trouver les résultats suivants :

- Le fichier *admin_a.php* :
  - Deux SQL injection dans la ligne 55 (figure 46)
  - Utilisation de fonction obsolète dans la ligne 55 (figure 46)
  - Reflected XSS dans la ligne 100 (figure 47), cette vulnérabilité a été causée par une variable qui a été infectée dans le fichier user.php dans la ligne 3 (figure 48)

```
55 $result=@mysql_query("SELECT * FROM utilisateur WHERE utilisateur=\"$username\" AND
     mot_de_passe=\"$passwordusername\"");
56
```

Figure 45 : extrait 1 du fichier admin_a.php.

```
 99   <div class="col-md-3">
100     <?= $nom_utilisateur ?>
101   </div>
```

Figure 47: 2e extrait du fichier admin_a.php

```
3  $nom_utilisateur = $_GET['nom_utilisateur'];
4
```

Figure 46:. Extrait du fichier user.php.

- Le fichier admin_b.php :
  - Ne contiens aucune vulnérabilité
- Le fichier user.php :
  - Ne contiens aucune vulnérabilité

➢ **Résultat de notre analyseur :**

Après avoir entré le chemin du répertoire, un scan a été effectué et voici le résultat :

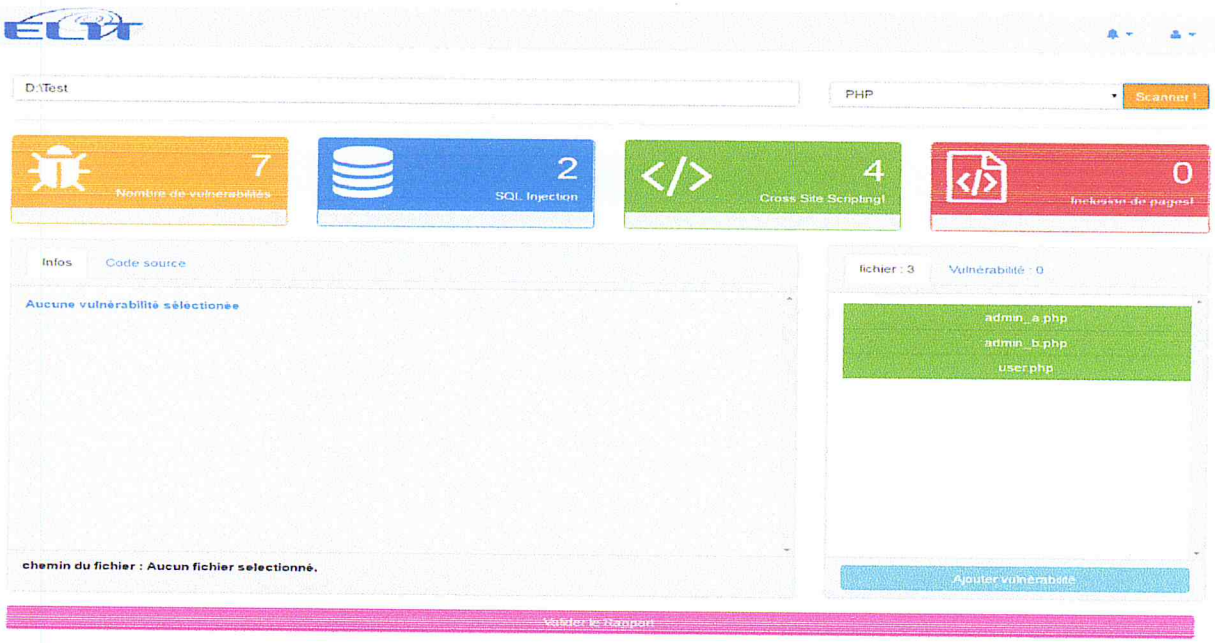


Figure 48 : Affichage du résultat du scan.

La figure 49 ci-dessus montre le premier résultat du scanne, on voit que notre analyseur a scanné trois fichiers en effet dans la figure 42 nous voyons qu'il y a quatre fichiers, cependant le quatrième fichier est un fichier HTML donc notre analyseur l'ignore automatiquement.

L'analyseur a aussi trouvé sept vulnérabilités alors que dans la partie du résultat attendu nous avions dit que le projet contient quatre vulnérabilités ce qui veut dire que notre analyseur a détecté trois faux positifs, nous allons voir par la suite pourquoi notre analyseur à trouver ces résultats et comment y remédier.

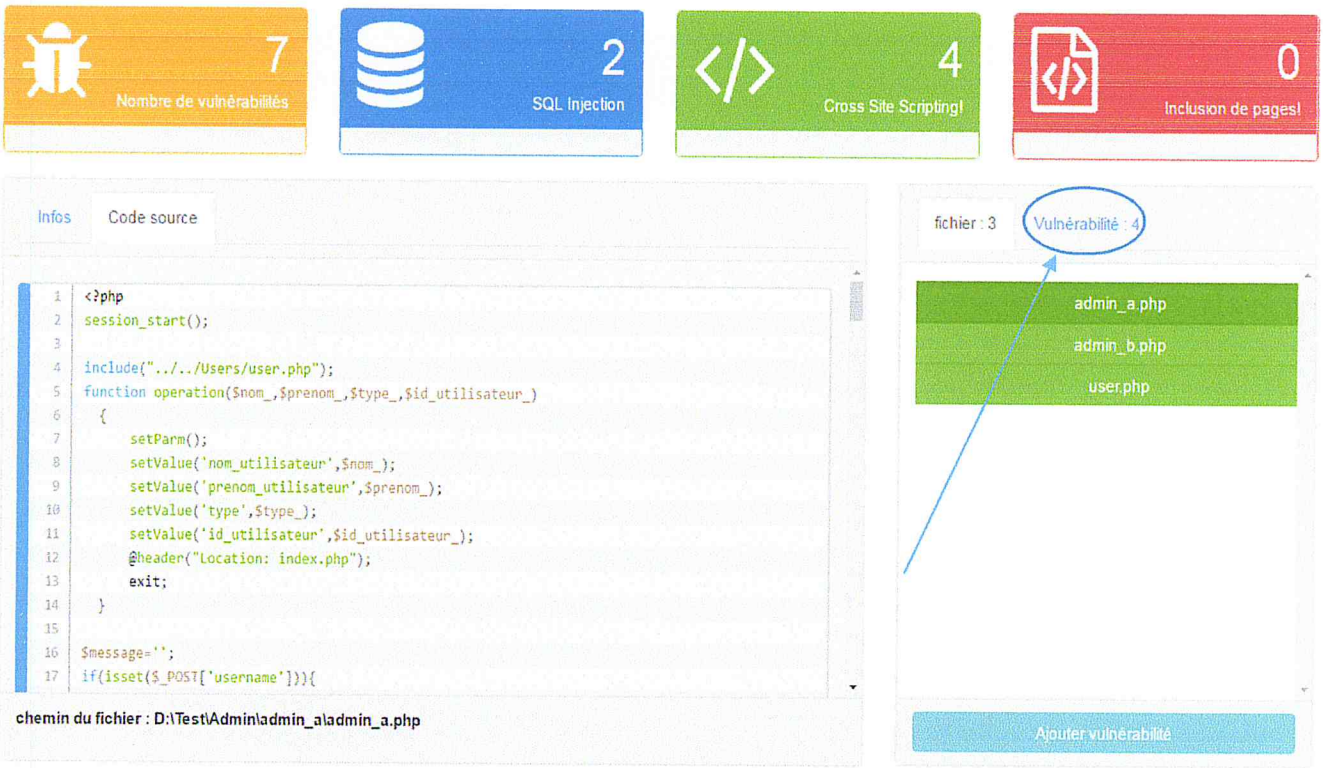


Figure 49 : résultat du scan du fichier.

Notre analyseur a détecté dans le fichier « admin_a.php » quatre vulnérabilités et c'est le résultat attendu reste à vérifier que ces quatre-là ne sont pas des faux positifs

Figure 50: Les vulnérabilités détectées dans le fichier admin_a.php.

Dans la figure précédente, notre analyseur a détecté les vulnérabilités attendues

- Deux vulnérabilités de type « Sql_injection » dans la ligne 55 (figure 51)
- Une utilisation de fonction obsolète ligne 55

Pour le premier fichier le résultat est correct ce qui implique que les faux positive détectés sont dans les autres fichiers, la figure ci-dessous montre le résultat du fichier

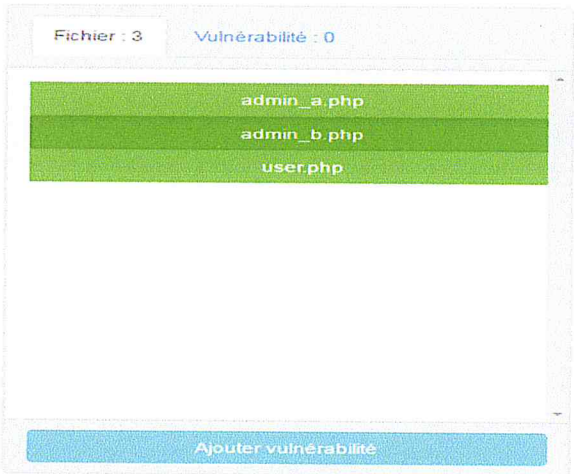


Figure 51 : Les vulnérabilités détectées dans le fichier admin_b.php

Aucune vulnérabilité détectée dans le fichier admin_b.php (figure 52), ce résultat est correct et correspond au résultat attendu, ensuite, notre analyseur a détecté dans ce fichier trois vulnérabilités, ces vulnérabilités sont des faux positifs, la figure suivante donne plus de détails sur ces derniers.

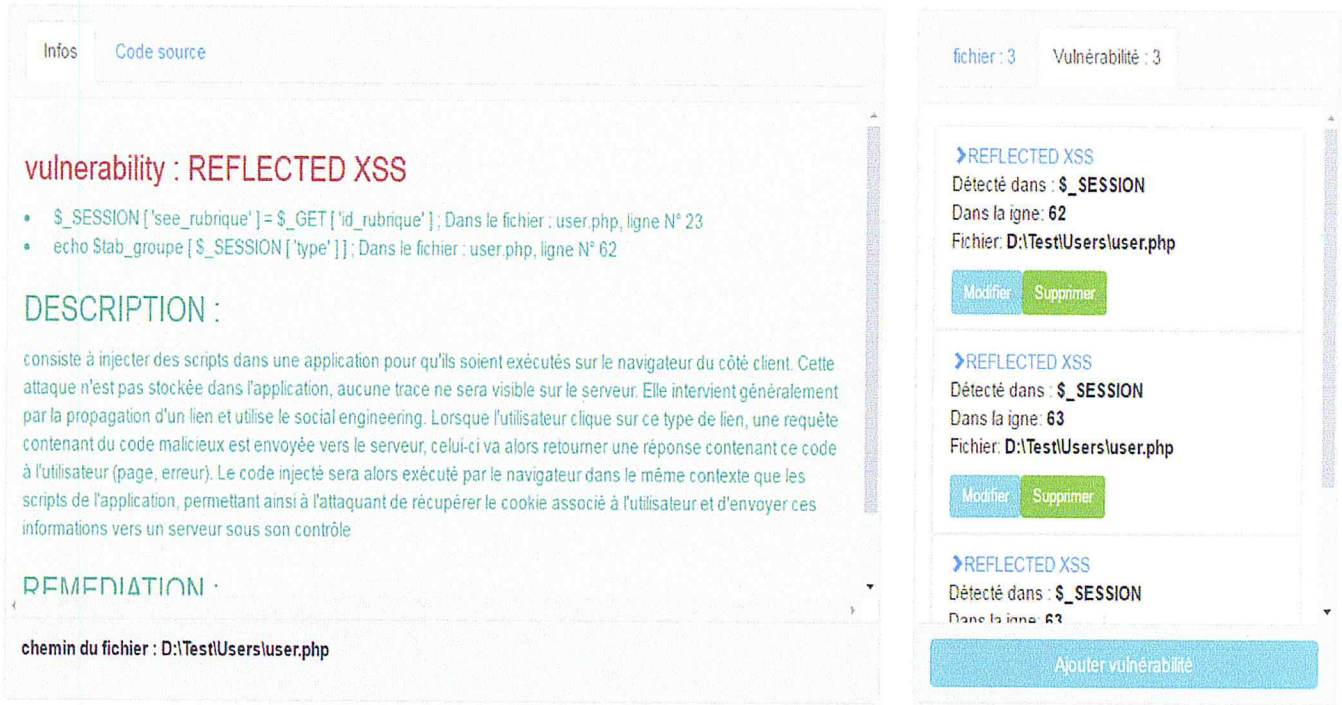


Figure 52: Les vulnérabilités détectées dans le fichier user.php

Dans la figure ci-dessus nous remarquons que la vulnérabilité détectée a été causée par la variable « $_SESSION »

- « $_SESSION » a été infecté par la variable « $_GET » dans la ligne 23 « `$_SESSION['see_rubrique']=$_GET['id_rubrique'];` »
- Après dans la ligne 63 on trouve qu'il y a une possibilité d'attaque XSS « `php echo $_SESSION['nom_utilisateur']` »

On constate que la variable « $_SESSION » est un tableau, et que la case avec l'index « see_rubrique » est infectée et non pas la case avec l'index « nom_utilisateur », mais l'analyseur a détecté une vulnérabilité car il a pris en considération seulement la variable « $_SESSION ».

Remarque : L'analyseur a détecté seulement une vulnérabilité car on n'a pas implémenté un modèle pour l'infection des tableaux.

Comme solution rapide et simple à ce problème l'auditeur peut supprimer les faux positifs détectés (figure 54) :

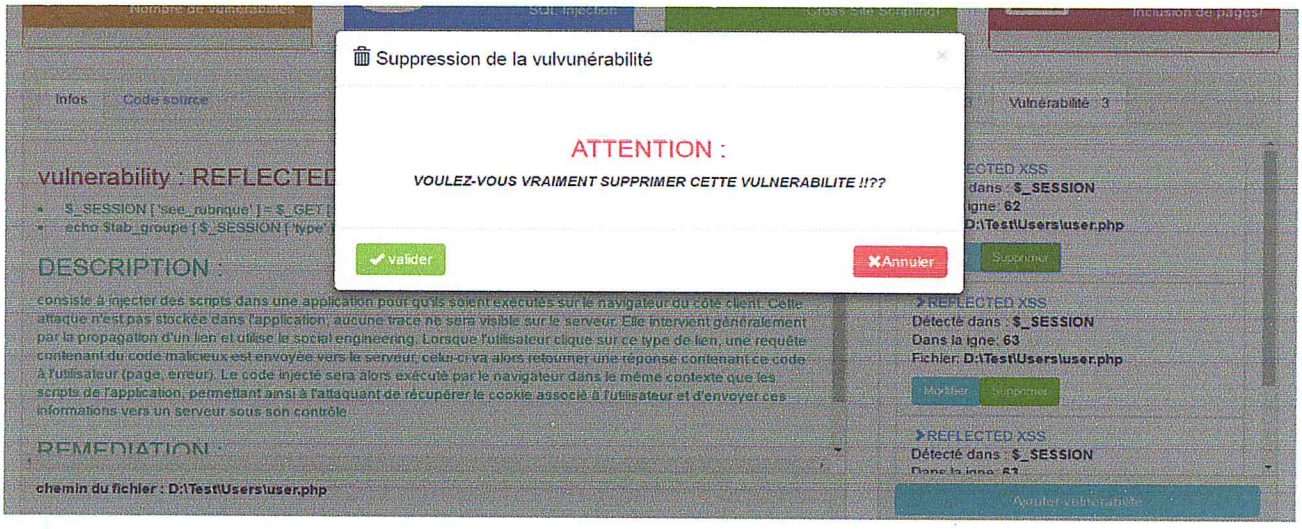


Figure 53: Suppression des faux positifs.

L'auditeur peut aussi ajouter des vulnérabilités non détectées par notre analyseur (faux négatif),

Figure 54: Ajout d'une vulnérabilité.

Comme montré dans les figures 40 et 41 nous avons ajouté une vulnérabilité appelée « vul » dans la code source de l'implémentation de notre bibliothèque, nous allons voir maintenant les interfaces qui représentent le résultat avant d'avoir ajouté la vulnérabilité et après l'avoir fait.

Dans la figure 56 qui suit, le fichier scanné ne contient aucune vulnérabilité donc aucun résultat n'est affiché. (Vulnérabilité : 0)

Figure 55: Résultat avant l'ajout de la vulnérabilité.

Dans la figure 57, nous remarquons que la vulnérabilité " vul " a été détectée, et cela après qu'elle ait été ajoutée à la liste des vulnérabilités répertoriées dans la bibliothèque.

Figure 56: Résultat après l'ajout de la vulnérabilité.

Après que l'auditeur ait vérifié tous les résultats, il peut valider ces résultats et les envoyer au client depuis le système actuel.

5 Validation du système proposé :

La phase de validation de notre travail s'est passée après avoir terminé notre stage pratique et cela par des experts compétents en matière d'audit et de sécurité de l'information.

La validation s'est déroulée en appliquant certains critères et exigences que l'entreprise voulait trouver dans notre solution proposée, l'expert a mis à l'épreuve notre système et les résultats suivants en découlent :

	Le système proposé	Observations
Langages scannés	PHP	Le système proposé scanne pour l'instant le langage de programmation PHP et cela comme il a été formulé dans notre demande initiale.
L'ajout d'autre langage	Oui	Le système proposé est doté d'une extensibilité au niveau des langages qu'il peut analyser et qui permet d'ajouter très facilement d'autres analyseurs et d'autres langages.
Fichiers scannés	Tout le répertoire	L'analyseur appartenant au système permet de façon très efficace de parcourir tous les fichiers du répertoire d'un projet qu'on voudrait auditer.
Validation des résultats par l'auditeur	Oui	Ce critère cité dans la demande initiale a été respecté et s'intègre complètement dans la politique d'audit de l'entreprise et ceci en permettant de garder toujours un droit de validation sur tout ce que le système proposé.
Taux de faux positifs et faux négatifs	Assez élevé	Malgré un taux élevé de faux positifs et de faux négatifs, le système semble répondre aux besoins, et cela en offrant la possibilité de supprimer ou d'ajouter des vulnérabilités par l'auditeur afin d'y remédier, cette fonctionnalité semble utile et très appréciée par les membres de l'équipe.

Temps de scanne	Rapide	Le temps de scanne est rapide mais dépend du volume du projet a auditer de plus nous tenons compte que le système parcoure tous les fichiers un à un ce qui rend le temps un peu plus lent dans certains cas.
La sauvegarde des résultats	Oui	Cette fonctionnalité est citée dans notre demande initiale et répond aux exigences, ce qui apporte une flexibilité lors de la phase d'audit, ainsi l'analyse se déroulera une fois et les informations sauvegardées pourront être consultée à n'importe quel instant

Tableau 18 : Rapport de validation du système proposé.

L'expert Adjailia Romaissa auditeur et chef de département de la sécurité des systèmes d'information au sein de la filiale d'ELIT a validé le travail effectué et cela en appliquant sur ce dernier plusieurs critères qui ont donné des résultats très satisfaisants, elle a aussi affirmé que même si le système proposé n'était pour l'heure qu'un prototype il répondait bel et bien aux besoins de l'entreprise et aux exigences énumérées au début de notre stage pratique. En mentionnant que toutes les mesures ont été prises pour garantir une extensibilité du système afin de le rendre adaptable à n'importe quel projet auditable.

6 Conclusion :

À l'issue de ce chapitre, nous avons vu les différentes illustrations que compose le système actuel de l'entreprise ainsi que celles de notre application en précisant les quelques fonctionnalités des différents utilisateurs, nous avons vu également l'environnement de travail tel que les outils de développement ainsi que les langages de programmation.

Bien que notre analyseur ne soit pour l'instant qu'un prototype et qu'il ne détecte que quelques types de vulnérabilités, nous pouvons dire que les objectifs fixés par ELIT ont été atteints, et cela car il permet de :

- Réaliser une analyse statique du code source
- Minimiser les faux négatifs (Possibilité d'ajouter et de modifier des vulnérabilités)
- Minimiser les faux positifs (Possiblité de supprimer des vulnérabilités)
- Permettre d'avoir une interface simple et facile pour effectuer l'opération d'audit.

Conclusion générale et perspectives :

Le présent projet a porté sur la réalisation d'un système d'audit code source qui a pour but de détecter les failles et les vulnérabilités concernées par le langage PHP, certaines peuvent être détecté grâce à l'analyse statique, d'autres grâces à l'analyse dynamique, testée dans les deux sens rend l'analyse complète, l'une des raisons les plus importantes à cela est que le test avec une seule technique laisse un risque résiduel, par conséquent les applications les plus critiques doivent être testés de préférence à l'aide des deux techniques, ajoutant à cela l'analyse manuelle qui diminue de façon considérable l'apparition de faux positifs.

L'objectif de ce travail était d'apporter une nouvelle approche dans la sécurisation des logiciels afin de permettre un gain de temps considérable et cela dans le but d'éliminer et/ou de minimiser les inconvénients recensés tout au long de l'étude de l'existant.

L'étude conceptuelle nous a permis de présenter notre solution qui comme nous l'avons spécifiée combine plusieurs techniques afin d'offrir un système d'audit code source efficace et robuste offrant des fonctionnalités simples, claires et totalement nouvelles, cette étude nous a aussi permis de spécifier les nouvelles fonctionnalités du système proposé qui viendront compléter et améliorer le système existant toujours dans le souci d'offrir un système d'audit complet et satisfaisant.

Bien que pour l'instant notre système n'est qu'un prototype, il offre cependant plusieurs approches et plusieurs fonctionnalités qui seront appréciées, sa robustesse et son extensibilité permettront d'enrichir et de compléter le système par d'autres fonctionnalités et techniques comme proposées dans les perspectives, ce qui permettra au système de rivaliser avec les meilleurs outils d'audit code source existant dans le marché actuel.

Les fonctionnalités offertes par le système proposé ont atteint les objectifs fixés. Toutefois, le système d'audit code source peut être sujet à des améliorations :

a) Au niveau de l'analyseur :
   1. L'ajout de nouveaux modèles et techniques qui permettront de minimiser le taux de faux positifs de façon drastique, ainsi que la détection des erreurs syntaxique.
   2. La détection de nouveaux types de vulnérabilités autre que les injections (buffer overflow, null pointer exception, etc)

b) Au niveau du système :
   1. L'ajout d'une fonctionnalité qui permettra d'ajouter de nouvelles vulnérabilités directement sur l'interface.

À l'issue de ce projet de fin d'études dont les objectifs nous ont été assignés par notre encadreur, nous pouvons dire que les travaux effectués ont abouti aux résultats attendus à savoir proposer un système d'audit code source, et que cela puisse servir et contribuer aux efforts de perfectionnement qu'offre l'outil informatique dans les progrès de maîtrise de la sécurité des systèmes d'information.

Bibliographie :

1. figuigui, Ayboub. Introduction à la sécurité. www.nbs-system.com. [Online] juillet 17, 2010. [Cited: mai 03, 2017.]

2. A view of 20th and 21st century software engineering Shanghai. Boehm, Barry. chine : s.n., May 20 - 28, 2006. ICSE.

3. Leclerc, Patrick. Stratégies OWASP pour développer et exploiter des applications sécuritaires. www.owasp.org. [Online] owasp, mars 16, 2016. [Cited: 04 15, 2017.] https://www.owasp.org/index.php/Quebec_City.

4. Jacopini, Denis. l'expert, les risques dus à la sécurité, les-statistiques-du-numerique-usages-risques-cybercriminalite. lenetexpert.fr. [Online] juilllet 17, 2015. [Cited: avril 20, 2017.]

5. Tassey, Gregory. The Economic Impacts of Inadequate Infrastructure for Software Testing. Gaithersburg, : U.S Department of Commerce, 2002. 7007.011.

6. Wikipedia. wikipedia. www.wikipedia.com. [Online] septembre 25, 2016. [Cited: avril 03, 2017.] https://fr.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology.

7. owasp. owasp. owasp.org. [Online] mars 13, 2017. [Cited: juillet 3, 2017.] https://www.owasp.org/index.php/Main_Page.

8. Veracode corporation. veracode. veracode.com. [Online] janvier 2016. [Cited: juillet 03, 2017.]

9. OWASP. OWASP Top ten 2013. owasp.org. [Online] janvier 2013. [Cited: juillet 3, 2017.] https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf.

10. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC): Tutorialpoint-simplyeasy learning. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC). 2017. Vols. p 1-5. ISBN.

11. Dewhurst, Ryan. Implementing Basic Static Code Analysis into Integrated Development Environments (IDEs) to Reduce Software Vulnerabilities. Newcastle : s.n., 2011/2012.

12. Bender RBT. Systems Development Life cycle: Objectives and Requirements. s.l. : bender rbt, 2013. ISBN.

13. McGraw, Garry. la sécurité applicative et son cycle de développement. synopsys.com. [Online] mars 31, 2004. [Cited: mai 04, 2017.] https://www.synopsys.com/blogs/software-security/software-security/.

14. Microsoft. software security. microsoft.com. [Online] microsoft corporation, 2015. [Cited: avril 25, 2017.] https://www.microsoft.com/en-us/SDL/process/training.aspx.

15. Driver, Mark, et al. Magic Quadrant for Integrated Software Quality Suites. s.l. : gartner, 2004. ISBN.

16. GARTNER. Critical Capabilities. new york : Gartner inc, 2015. ISBN.

17. audry, Benoit. Test dynamique université rennes. teste dynamique.people.rennes.inria.fr . [Online] janvier 2015. [Cited: avril 15, 2017.]

18. baptiste, Philippe. L'héritage d'Alan Turing. cnrs.fr. [Online] mars 2015. [Cited: juillet 03, 2017.] http://www.cnrs.fr/fr/pdf/jdc/Turing.pdf.

19. OWASP. Introduction au code review. owaps.org. [Online] mars 2016. [Cited: avril 26, 2017.] https://www.owasp.org/index.php/Code_Review_Introduction.

20. université mlv. Definitions de faux positifs et faux négatifs. [Online] novembre 2015. [Cited: juillet 3, 2017.] http://igm.univ-mlv.fr/~dr/XPOSE2007/plebacco_ids/B_faux_.html.

21. Frederic, pascal. la méthode grep. www.eila.univ-paris-diderot.fr. [Online] mars 2012. [Cited: juillet 03, 2017.] ttp://www.eila.univ-paris-diderot.fr/_media/user/alexandra_volanschi/linux/grep.pdf.

22. Université laval. L'analyse lexicale. ift.ulaval.ca. [Online] décembre 2014. [Cited: juillet 03, 2017.] http://www2.ift.ulaval.ca/~dadub100/cours/A15/3101/slides3.pdf.

23. Christian, Charras and Thierry.Lecroq. Le string matching. université MLV. [Online] avril 1997. [Cited: juillet 3, 2017.] http://www-igm.univ-mlv.fr/~lecroq/string/node2.html.

24. NKENLIFACK, Marcellin Julius. Le flux de données. foad-mooc.auf.org. [Online] decembre 2009. [Cited: juillet 03, 2017.] http://www.foad-mooc.auf.org/IMG/pdf/IngenierieSystLogiciel_NK_2009.pdf.

25. Barbosa, Edgar. The taint analysis. istate.edu. [Online] mars 2009. [Cited: juillet 03, 2017.] http://web.cs.iastate.edu/~weile/cs513x/5.TaintAnalysis1.pdf. ISBN.

26. Djamila, Mokeddem and Hafida, Belbachir. Utilisation des méthodes d'apprentissage ensembliste dans le Datamining distribué. webreview.dz. [Online] 2007. [Cited: juillet 03, 2017.] http://www.webreview.dz/IMG/pdf/mokkedam.pdf.

27. Schobbens, PY. Théorie des langages : Syntaxe et sémantique, Syntaxe et sémantique. gaudry.be. [Online] janvier 2010. [Cited: juillet 3, 2017.] http://www.gaudry.be/langages-analyse-syntaxique-ast.html.

28. Feautrier, Paul. Static Single Assignment. ens-lyon.fr. [Online] octobre 2014. [Cited: juillet 3, 2017.] http://perso.ens-lyon.fr/paul.feautrier/ssa.pdf.

29. Antoine, Miné. Analyses de pointeurs, TAS : Typage et analyse statique. master.ufr-jussieu.fr. [Online] mars 24, 2016. [Cited: juillet 3, 2017.] https://www-master.ufr-infop6.jussieu.fr/2015/spip.php?action=acceder_document&arg=19832&cle=2a0027764dc016534780320c112a1d22ee272506&file=pdf%2Fcours-13_ok.pdf.

30. OWASP. sql injection. owasp.org. [Online] janvier 2016. [Cited: juillet 03, 2017.] https://www.owasp.org/index.php/SQL_Injection.

31. —. crosse site scripting (XSS). owaps.org. [Online] 2016. [Cited: juillet 03, 2017.] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

32. owasp. Injection de commande. owasp.org. [Online] 2016. [Cited: juillet 3, 2017.] https://www.owasp.org/index.php/Command_Injection.

33. wamp server site officiel. présentation du serveur WAMP. wampsserver.com. [Online] [Cited: mai 4, 2017.] http://www.wampserver.com/?lang=fr.

34. Welling, Luke and Thomson, Laura. PHP & MYSQL. paris : 4 eme edition, 2009. isbn.

35. site officiel phpmyadmin. présentation de la plate-forme de la base de données. phpmyadmin.net. [Online] 2016. [Cited: mai 15, 2017.] http://www.phpmyadmin.net.

36. Site web officiel de l'editeur de texte sublime 3. présentation et telechargement de l'outil sublime-text. sublime-text.fr. [Online] janvier 2017. [Cited: mai 15, 2017.] http://sublime-text-3.fr.uptodown.com/.

37. Johann, Pardanaud. présentation de ajax. openclassrooms.com. [Online] avril 17, 2017. [Cited: mai 15, 2017.] https://openclassrooms.com/courses/dynamisez-vos-sites-web-avec-javascript/l-ajax-qu-est-ce-que-c-est.

38. MIT, et al. qu'est-ce que le HTML. w3.org. [Online] 2015. [Cited: mai 31, 2017.] http://www.w3.org/standards/webdesign/htmlcss.

39. Budd, Andy, Mall, Cameron and Clisso, Simon. Maîtrise des CSS. s.l. : 2eme edition, 2009. ISBN.

40. Templier, Thierry and Gougeon, Arnaud. JavaScript pour le web 2.0. s.l. : Edition Eyrolles, 2007.

41. Chevalli, Maurice. Decouvrez le framework PHP LARAVEL. s.l. : Eyrolles, 2016. ISBN.

42. Skyorc, Bruno. les framework PHP les plus populaires. sitepoint.com. [Online] 2015. [Cited: juin 16, 2017.] https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/.

43. Roques, Pascal. Le Cahier du programmeur. s.l. : Eyrolles, 2002.