



MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE
LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et de l'Informatique
Département de Mathématiques et d'Informatique
Filière : Informatique

MÉMOIRE DE FIN D'ETUDES
Pour l'Obtention du Diplôme de Master en Informatique
Option : **Ingénierie des Systèmes d'Information**

THÈME :

Développement d'un Web-Robot pour Analyser les
Clés Publiques et Certificats

Etudiant(e) : « *BOUHADJLA Houria* »

Encadrant(e) : «Dr. FILALI F.Z. »

Année Universitaire 2016/2017

Résumé

Un robot d'indexation est un programme d'exploration utilisé par le web pour rechercher et indexer dans les bases de données les nouveaux sites, les nouvelles pages et les pages modifiées. Les robots d'indexation se rendent sur les pages, les lisent, et suivent les liens hypertextes afin de collecter les informations des pages sous forme de métadonnées, pour ordonner les résultats de recherche en fonction de leur pertinence.

Cependant, avec le développement croissant d'Internet, les applications web sont devenues de plus en plus vulnérables et exposées à des attaques malveillantes pouvant porter atteinte à des propriétés essentielles telles que la confidentialité, l'intégrité ou la disponibilité des systèmes d'information. Pour faire face à ces malveillances, il est nécessaire de développer des mécanismes de protection et de test (tel que le crawler web) qui soient efficaces. De plus, le robot proposé doit être silencieux pour ne pas gêner les fournisseurs de services, mais doit gérer la priorité des services dans un délai raisonnable.

Dans ce travail, nous essayons de proposer une solution qui consiste à concevoir et à développer un robot web pour récupérer les clés publiques et certificats des services Internet afin de remédier aux problèmes cités.

Mots-clés : robot, web, crawler, Certificats, Clé publique, service web, sécurité des services web, scanner de sécurité d'applications web, Algorithmes de web crawler.

Abstract

A robot of indexing is an exploration program used by the web to search and index in the databases for new sites, new or modified web pages.

The robots of indexing visits the pages, read them, and follow the hyperlinks in order to collect information about pages in the form of metadata, to sort search results according to their relevance.

However, with the growth of the Internet, the web applications became increasingly vulnerable and exposed to malicious attacks. Thus, this reached us to the use of essential properties such as the confidentiality, the integrity or the availability of information systems.

To prevent this, it is necessary to develop effective protection and vulnerability tests mechanisms (such as web crawlers). Moreover, the robot purpose must be quiet and don't obstruct the service providers, but must manage the priority of the services within a reasonable delay.

In this work, we try to propose a solution that consists of designing and developing a web-robot based on security mechanisms such as public keys and service provider's certificates in order to resolve the mentioned problems.

Keywords: robot, web, crawlers, certificates, public key, web service, web service's security, scanner of web applications' scanner, web crawlers' algorithms.



Remerciements

*Avant tout, je remercie **DIEU** le tout puissant de m'avoir accordée la force et le courage pour réaliser ce modeste travail, atteindre mon but et réaliser ainsi un rêve.*

Je tiens à exprimer toute ma reconnaissance à mon encadreur de mémoire Madame "Filali Fatima Zohra". Je la remercie de m'avoir encadré, orienté, aidé et conseillé.

Je tiens également à remercier les membres du jury qui ont bien voulu accepter d'examiner ce travail.

Je souhaite remercier mes très chers parents qui tiennent une place immense dans mon cœur, et qui sans eux je ne serais jamais ce que je suis aujourd'hui. Il m'est impossible de trouver des mots pour dire à quel point je suis fière d'eux, et à quel point je les aime.

Que Dieu vous bénisse !

Je suis redevable d'une éducation dont je suis fier.

Enfin, d'un point de vue personnel je remercie tous mes amis et proches qui ont été à mes côtés pendant ces dernières années d'études, Je voudrais exprimer ma reconnaissance envers eux qui m'ont apporté leur support moral et intellectuel tout au long de ma démarche.

Un grand Merci à tous ceux que j'ai oubliés et qui se reconnaîtront.



BOUHADJLA Houria





Je dédie cette thèse ...

A mon père, Aucune dédicace ne saurait exprimer l'amour, l'estime, le dévouement et le respect que j'ai toujours eu pour vous.

tu es une vraie école de la vie, je ne cesse d'apprendre tous les jours avec toi.

tu as su m'entourer d'attention, m'inculquer les valeurs nobles de la vie, de l'honnêteté et de la responsabilité.

Ce travail est le fruit de tes sacrifices que tu as consentis pour mon éducation et ma formation.

A la plus douce et la plus merveilleuse, A une personne qui a tout donné sans compter " ma très chère mère " la source de tendresse et l'exemple du dévouement qui n'a pas cessé de m'encourager et de prier pour moi.

Ta prière et ta bénédiction m'ont été d'un grand secours pour mener à bien mes études.

Je te dédie ce travail en témoignage de mon profond amour.

Puisse Dieu, le tout puissant, te préserver et t'accorder santé, longue vie et bonheur afin que je puisse vous rendre un minimum de ce que je vous dois.

Des mots ne pourront jamais exprimer la profondeur de mon respect, ma considération, ma reconnaissance et mon amour éternel.

A ma très chère sœur Sabrina : Mon amour pour vous est sans limite.

J'implore DIEU qu'il vous apporte bonheur, amour et que vos rêves se réalisent.

A mes frères Fayçal et Farouk : Je vous souhaite pour les deux une longue vie pleine de réussite, de santé et de bonheur pour accomplir tous vos objectifs.

Ma belle famille Vous êtes la joie de ma vie, Que Dieu vous garde et vous protège.

A mes chers oncles (Mohamed, Saïd, Hamid, Azzedine, Ibrahim, Ismail) et leurs épouses, et mes tantes ("et bien sûr sans oublier leurs enfants").

Veillez trouver dans ce travail l'expression de mon respect le plus profond et mon affection la plus sincère.

A mes chères amies .

En souvenir de notre sincère et profonde amitié et des moments agréables que nous avons passés ensemble.

Veillez trouver dans ce travail l'expression de mon respect le plus profond et mon affection la plus sincère

A tout mes collègues sans exception.

Merci à tous

Bouhadjla Houria



Table des matières

Résumé	i
Abstract	i
Table des matières	v
Liste des figures	ix
Liste d'abréviations	x
Introduction Générale.....	1
Chapitre 1 : Etat de l'art	3
I. Les Robots d'indexation	4
1. Introduction	4
2. Définition de Crawler	4
3. Principe de crawler.....	4
4. Fonctionnement	4
5. Architecture générale	5
5.1. Architecture de 2-modules.....	5
5.2. Architecture de 4-modules.....	5
5.3. Architecture logicielle	6
6. Problèmes à considérer pour les web crawler.....	7
6.1. Taux de couverture du web et évolutivité	7
6.2. Taux d'actualisation et de téléchargement des pages web	7
6.3. La performance	7
6.4. Recherche de contenu manquant ou caché	8
7. Stratégies des web crawlers	8
7.1. Algorithme de parcours en largeur	8
7.2. Algorithme de parcours en profondeur	9
7.3. Algorithme de Pagerank	9
7.4. Algorithme génétique	10
7.5. Algorithme de classification naïve bayésienne.....	10
7.6. Algorithme HITS.....	10
II. Sécurité des services web	11
1. Définition	11
2. La sécurité des services web	11

3.	Besoins de sécurité des services web	11
3.1.	Autorisation	11
3.2.	Confidentialité	12
3.3.	Authentification	12
3.4.	Intégrité	13
4.	Protection des données (confidentialité et Authentification)	13
4.1.	Chiffrement à clé publique et privée	13
4.2.	Certificats	13
4.3.	SSL	14
5.	Autres besoins de sécurité à considérer	15
5.1.	Traçabilité et audit	15
5.2.	Utilisation de protocoles standards	15
6.	Scanner de sécurité des applications web	16
Chapitre 2 : Conception et modélisation		17
Introduction		18
1.	Présentation de la solution proposée	18
1.1.	Description de l'existant	18
1.1.1.	Principe des outils de détection de vulnérabilités	18
1.1.2.	Analyse critique des scanners de vulnérabilités web	20
1.2.	Solution proposée	21
1.3.	Choix du modèle proposé	21
1.3.1.	Crawling	23
1.3.2.	Scanning	23
2.	Spécification des besoins	23
2.1.	Collecte d'informations	24
2.1.1.	Crawler le contenu effacé ou caché :	24
2.1.2.	Récupérer des métafichiers du serveur web pour des informations supplémentaires	24
2.1.3.	Examiner les commentaires et les métadonnées des pages web	25
2.1.4.	Vérifier la plateforme de l'application web	25
2.1.5.	Tester les définitions des rôles	26
2.1.6.	Identifier les points d'entrée de l'application	26
2.2.	Transmission Sécurisée :	27
2.2.1.	Vérifier les protocoles SSL /TLS	27

2.3. Authentification.....	28
2.3.1. Tester les règles de qualité des mots de passe.....	28
2.3.2. Tester de l'énumération des utilisateurs	28
2.3.3. Tester le contournement du schéma d'authentification	28
2.3.4. Tester la transmission d'informations d'authentification sur un canal crypté	29
2.3.5. Tester par force brute	29
2.4. Cryptographie	30
2.4.1. Vérifier si les données à crypter ne le sont pas	30
2.4.2. Vérifier l'utilisation d'algorithmes faibles.....	30
2.5. Validation des données	31
2.5.1. Test pour injection SQL.....	31
2.5.2. Test pour XSS (Cross Site Scripting) réfléchi.....	32
3. Conception	32
3.1. Langage UML.....	32
3.1.1. Définition UML.....	33
3.1.2. Les diagrammes UML	33
3.2. Modélisation dynamique	34
3.2.1. Diagramme de séquence	35
3.2.2. Diagramme de collaboration :.....	36
3.2.3. Diagrammes d'activités.....	37
3.3. Modélisation statique.....	38
3.3.1. Diagrammes de classes.....	38
3.3.2 Architecture de l'application.....	38
Conclusion	40
Mise en œuvre et implémentation	41
Introduction	42
1. Environnement du travail	42
1.1. Environnement Matériel	42
1.2. Environnement logiciel	42
1.2.1. Python.....	42
Pourquoi python ?.....	42
1.2.2. Java et IDE NetBeans	42
1.2.3. Jython	43

2. Architecture générale du système	43
2.1. Le Crawling	43
2.2. Le scanner	43
2.3. Les résultats	44
3. implémentation des composants du système	44
Conclusion	49
Conclusion générale	50
Références Bibliographiques	51

Liste des figures

Figure 1. Architecture de 2-modules. [3]	5
Figure 2. Architecture de 4-modules [3]	6
Figure 3. Composants logiciels d'un crawler [3]	7
Figure 4: Parcours en largeur [7].....	9
Figure 5. Parcours en profondeur [8]	9
Figure 6. Extraction et recherche des points de vulnérabilités	22
Figure 7. Diagramme de séquence « Détection de vulnérabilités».	35
Figure 8. Diagramme de collaboration «Test de vulnérabilités».	36
Figure 9. diagramme d'activités «Evaluation de vulnérabilités».....	37
Figure 10. Architecture générale de notre application	39
Figure 11. Interface de Netbeans IDE 8.2	43
Figure 12. Lancement du Test de vulnérabilités	44
Figure 13. Fin du test de vulnérabilités	44
Figure 14. Les résultats du test	45
Figure 15. Résultats du Crawling	45
Figure 16. Interface Graphique de l'application	46
Figure 17. Contenu du fichier.....	47
Figure 18. Résultat Final	48
Figure 19. Affichage de l'exécution.....	49
Figure 20. Histogramme des points des sécurités	49

Liste d'abréviations

URL	Uniform Resource Locator
FIFO	First in, first out
HITS	Hyperlink-Induced Topic Search
HTML	Hypertext Markup Language
XML	Extensible Mark-up Language
JSON	JavaScript Object Notation
HTTP	Hyper Text Transfer Protocol
SAML	Security assertion markup language
WS-Security	sécurité des services web
SOAP	protocole Simple Object Access
(AC ou CA)	Les autorités de certification
SSL	signifie Socket Secure Layer
TLS	Transport Layer Security
XML-RPC	XML- Remote procedure call
REST	Représentationnel State Transfer
WSDL	Web Services Description Language
UDDI	Universal Description, Discovery and Integration
WASSEC	Web Application Security Scanner Assessment Criteria
OWASP 4	Open Web Application Security Project
WASC5	Web Application Security Consortium
XSS	Cross Site Scripting
REP	Robots Exclusion Protocol
SSL	Secure Sockets Layer
TLS	Transport Layer Security
SGBD	Système de gestion de base de données
UML	Unified Modeling Language

Introduction Générale

Centre d'intérêt

La taille et la dynamique du web nécessitent la maintenance et la mise à jour continue des systèmes d'information basés sur le web. Le web est une collection immense de pages (nœuds contenant de l'information) reliées entre elles avec les hyperliens.

La fonction qui consiste à visiter sans relâche et de manière automatisée les pages du web s'appelle le crawling. Elle est réalisée par un crawleur, appelé aussi spider ou bot. Tous ces termes désignent en fait le même logiciel : le robot d'indexation.

Son objectif est de récupérer des informations, pour les filtrer, classifier, indexer, mettre dans des bases de données, et donc de les exploiter par la suite.

Le crawl a commencé par un "mapping" du web et la connexion des sites entre eux. Un robot d'indexation était originalement utilisé par les moteurs de recherche pour découvrir de nouvelles pages et les indexer. Ils sont également utilisés dans d'autres cas de figure notamment dans la sécurité pour tester si un site est vulnérable ou non.

D'autre part, les services web sont devenus une technique incontournable pour construire des systèmes distribués faiblement couplés. Ainsi, il est primordial de fournir une bonne sécurité pour le service web. Les spécifications de sécurité incluent : l'autorisation, la confidentialité, l'authentification, l'intégrité...

Contexte et Problématique

La sécurité des applications Web est un problème désormais récurrent. Il existe de nombreuses attaques qui se focalisent sur les applications Web puisque ces derniers sont souvent mal configurés ou mal maintenus ce qui permet à des utilisateurs mal intentionnés de profiter des failles de sécurité de ces applications.

La détection de ces failles d'une part et des intrusions d'autre part trouve alors tous ses justificatifs en ce sens qu'à défaut de prévenir et protéger de façon sûre contre les attaques, les détecter permet néanmoins de limiter les dégâts et de réagir à temps.

Il est donc nécessaire d'auditer régulièrement les applications Web pour vérifier la présence de vulnérabilités exploitables et ceci peut être réalisé notamment par des scanners de vulnérabilités Web.

Enfin, il est également fondamental de pouvoir détecter les tentatives d'attaques à l'aide des techniques de protection pour les applications Web et nous nous focalisons principalement sur les scanners de vulnérabilités Web et l'utilisation des crawlers pour la détection et ensuite par la spécification des différents besoins de sécurité pour notre application.

Objectifs

L'objectif de notre travail est de développer un robot d'indexation pour détecter et tester la sécurité des services web.

Pour cela, nous décrivons les principes des scanners de vulnérabilités et nous analysons les limites de ces outils en se fondant sur de classifications pour les vulnérabilités et les attaques web les plus répandues. Parmi ces communautés, nous citons OWASP 4 (Open Web Application Security Project) et WASC5 (Web Application Security Consortium).

Organisation des documents

Ce projet s'intéresse au développement d'un robot d'indexation pour analyser et détecter les vulnérabilités d'applications web. Ce thème est organisé en trois chapitres :

Le chapitre 1 est constitué de deux parties à savoir :

La première partie est un état de l'art sur les robots d'indexation. Dans la deuxième partie, nous passerons en revue les spécifications de sécurité des services web. Dans cette partie, nous nous intéressons aux besoins de sécurité des services web et les protections possibles des données.

Le chapitre 2 de ce mémoire se présente et comprend trois parties comme suit :

Dans la première partie, nous présenterons quelques techniques utilisées par les scanners de vulnérabilités Web existants.

Dans la deuxième partie, nous présentons la solution proposée. Premièrement nous commencerons par une description de l'existant : nous analysons brièvement les limites des outils présentés. Deuxièmement nous expliquerons la solution et le modèle proposé qui vise à atteindre notre objectif. Nous proposons une approche permettant la détection de différents types de vulnérabilités Web, en utilisant deux étapes pour traiter le problème :

- le crawling, pour scanner les adresses URL afin de tester du contenu Web malveillant et des vulnérabilités de sécurité communes.
- le scanning consiste à tester chaque partie des composants de l'application pour un ensemble appliqué de vulnérabilités.

Troisièmement, nous discuterons de la conception de notre solution, nous détaillerons les outils, langages et diagrammes utilisés lors du processus de conception.

Dans le chapitre 3, nous montrerons l'implémentation de notre travail et les résultats obtenus à travers des captures d'écran.

Enfin, nous terminerons ce mémoire par une conclusion générale qui récapitule les travaux réalisés et fait le point sur un ensemble de perspectives envisagées et les orientations de recherches futures à partir de notre travail.

Chapitre 1 : Etat de l'art

I. Les Robots d'indexation

1. Introduction

Le web, qui se caractérise par sa grande taille et sa nature dynamique, nécessite un entretien constant spécifiquement pour les systèmes de recherche d'information. Si en revenons aux années 90, le volume du web était de quelques millions de pages passant à des milliards à l'heure actuelle, Google répertorie environ 1000 milliards de pages sur le web.

Les robots d'indexation facilitent le processus de recherche d'informations en suivant les liens hypertextes dans les pages web pour télécharger automatiquement et mettre à jour de nouvelles pages web afin de les stocker dans une base de données ou indexés pour répondre aux demandes (requêtes) des utilisateurs. [1]

2. Définition de Crawler

Les robots web (aussi connu comme crawleurs ou Spiders) sont des programmes qui reflètent le web automatiquement. Les moteurs de recherche comme Google les utilisent pour indexer le contenu web. Ils peuvent être utilisés à différents buts, tels que :

- Dans les moteurs de recherche, telles que pages d'index web.
- Pour l'archivage web (un grand ensemble de pages Web sont régulièrement rassemblées et archivées).
- Pour le Data Mining. Les pages web analysées en fonction des propriétés statistiques afin d'être utilisées pour l'analyse des données.

Le crawleur commence par une liste d'URL à visiter, appelée les graines. En visitant l'URL, le crawler identifie tous les liens hypertextes dans la page et les ajoute à la liste des URL à visiter, appelée la frontière crawl. La frontière URL est visitée, selon un ensemble de règles. [1]

3. Principe de crawler

Un Crawler extrait des éléments jugés significatifs et pertinents pris en compte afin de fournir la base de mots-clés liés à l'analyse de page. Comme le spider détecte des liens vers d'autres pages, il se souvient de la base de données contenant les adresses restantes à analyser. Une fois que la page est analysée, le spider recherche dans sa base de données la page suivante à visiter.

Un crawleur est capable de voyager de lien en lien, de page et page, et de site en site sans aucune intervention humaine. Il utilise le concept de lien hypertexte et les fondations du web.

4. Fonctionnement

Pour comprendre comment fonctionne un crawleur, il est nécessaire de rappeler le fonctionnement du web lui-même. Cela peut être vu comme un ensemble de ressources, chacune identifiée par une adresse unique appelée URL ou adresse web. Dans le cas où la ressource considérée est une page web, celle-ci pourra être affichée dans un navigateur web.

Une page web est susceptible de contenir un certain nombre de liens vers d'autres ressources de chaque page sur l'Internet.

En théorie, l'algorithme utilisé pour mettre en œuvre crawler est très simple. Celui-ci n'utilise qu'une seule structure de données : une collection d'adresses web bien connues.

La structure ne contient au départ qu'un petit nombre d'adresses web appelées candidats initiaux. Le crawler télécharge chacune des pages web et identifie les liens qu'elle contient.

Pour chaque lien, si l'adresse est inconnue, elle est ajoutée à la liste. L'algorithme choisit alors un nouvel ensemble d'adresses à visiter parmi ceux qui avaient été découverts et le processus se répète. [2]

5. Architecture générale

Il faut qu'un crawler ait une architecture optimisée qui récupère plusieurs pages par seconde pendant un court laps de temps. Mettre en œuvre un système à haute performance qui peut télécharger des centaines de millions de pages, présente un certain nombre de difficultés dans la conception de système, l'entrée-sortie et l'efficacité du réseau, la robustesse et l'administration.

5.1. Architecture de 2-modules

L'architecture d'un crawl contient deux composants principaux : **Downloader** et **scheduler**.

- Le **scheduler** calcule le score des pages pour déterminer l'ordre de traitement des pages.
- Le **downloader** télécharge les pages, les analyse, extrait les nouveaux liens et maintient la structure des liens.



Figure 1. Architecture de 2-modules. [3]

5.2. Architecture de 4-modules

L'architecture de 2-module, présente quelques problèmes. Premièrement lorsque le scheduler travaille sur le graphe du web, le graphe de web ne peut pas être changé. Ainsi, la tâche d'analyse de la page peut être longue. De même, lors de l'analyse, le graphe web est

occupé. La solution à ce problème est d'analyser toutes les pages de téléchargement en même temps, collecter les liens, puis les ajouter à la collection. Un autre problème est dans l'organisation du module downloader. La tâche d'analyse d'une page peut être très coûteuse alors que la tâche de téléchargement ne nécessite que de bonne connexion de réseau et les disques durs rapides. Les téléchargements des pages sont souvent faits par des processus en parallèle, donc chaque tâche de téléchargement est très légère.

L'architecture de 4-modules est proposée par Carlos Castillo pour améliorer la performance des crawls du web. Elle se compose de 4 modules suivants :

- **Manager** : calcule le score des pages et génère la liste de K URLs pour le téléchargement dans un cycle de traitement.
- **Harvester** : télécharge les pages
- **Gather** : analyse les pages et extrait les liens
- **Seeder** : maintien la structure de liens. [3]



Figure 2. Architecture de 4-modules [3]

5.3. Architecture logicielle

On peut modéliser les différents composants d'un crawler de la façon suivante :

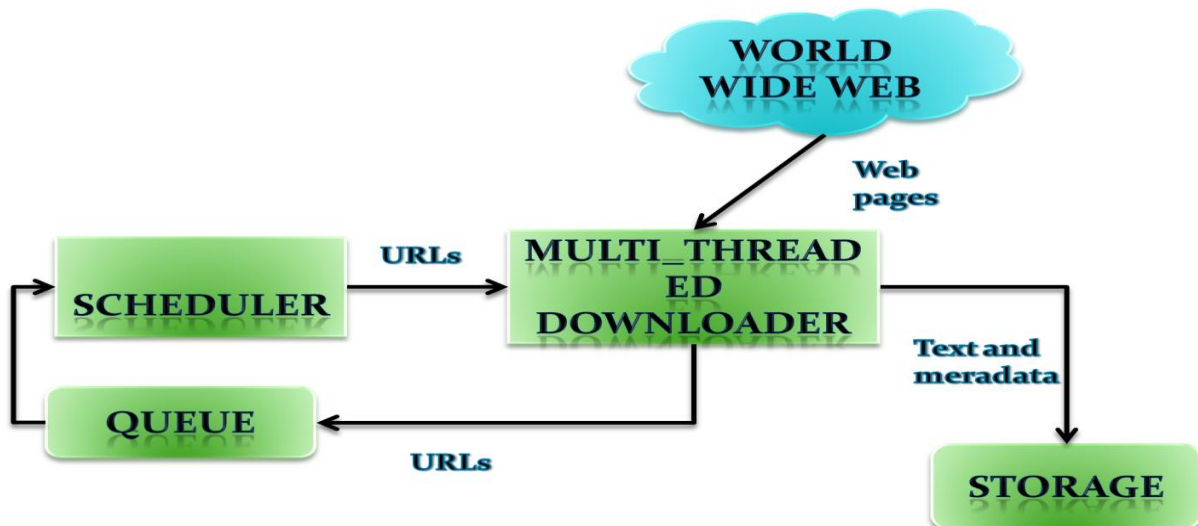


Figure 3. Composants logiciels d'un crawler [3]

Cette représentation est à un niveau élevé dans une certaine mesure, c'est la base pour créer un crawler extensible et efficace. L'utilisation d'un ordonnanceur (scheduler) a donné la priorité à des URL à visiter et permet de mieux gérer les différents temps morts entre le téléchargement des pages. [2]

6. Problèmes à considérer pour les web crawler

6.1. Taux de couverture du web et évolutivité

Le taux de couverture du web et évolutivité représente l'ensemble de pages souhaitées que le crawler acquière avec succès.

La comparaison du nombre de ressources découvertes par un crawler sur un site web particulier, avec les ressources cherchées par tous les clients de ce site, fournit une évaluation de l'exhaustivité de la visite de ce crawler sur le site particulier.

Une couverture d'un site web peut être définie comme le rapport des demandes distinctes d'URL a publié par une chenille au-dessus des demandes distinctes d'URL fournies par tous les web client au site actuel, sur le même laps de temps. [4]

6.2. Taux d'actualisation et de téléchargement des pages web

Beaucoup de pages consistent en des informations qui sont mises à jour sur une base quotidienne, hebdomadaire ou mensuelle. Le robot d'indexation doit télécharger ces pages et les mettre à jour dans la base de données pour fournir des informations à jour aux utilisateurs. Le processus de crawling devient lent et exerce une pression sur le trafic Internet si ces pages sont en grand nombre. Ainsi, un problème majeur est de développer une stratégie qui gère ces pages. [5]

6.3. La performance

La performance est généralement mesurée par le pourcentage de pages téléchargées qui sont pertinentes pour le sujet.

Cette mesure est appelée « taux de récolte ». Le taux de récolte peut être ajusté (en utilisant un seuil plus élevé) pour mesurer la capacité de l'outil de chenilles à télécharger des pages très pertinentes pour le sujet.

La performance d'un crawler est calculée comme le nombre moyen de résultats positifs sur tous les sujets. [6]

6.4. Recherche de contenu manquant ou caché

Une version mise en cache d'une page web est une copie de la page telle qu'elle était dans le dernier passage du web crawler.

Cette problématique consiste à concevoir un cache efficace qui a un taux de succès élevé et qui surmonte également les contraintes du réseau.

7. Stratégies des web crawlers

Nous présentons dans cette section un certain nombre d'algorithmes de recherche des crawlers qui sont suggérés dans la littérature.

La plupart de ces algorithmes sont des variantes du schéma best-first. La différence est dans les heuristiques qu'ils utilisent pour marquer les URL non visitées.

7.1. Algorithme de parcours en largeur

L'algorithme de parcours en largeur fonctionne par niveau, l'algorithme démarre à l'URL racine et recherche l'URL de tous les voisins au même niveau. Si l'URL recherchée est trouvée, la recherche se termine. Si ce n'est pas le cas, la recherche se poursuit et le processus est répété jusqu'à ce que le but soit atteint.

Lorsque toutes les URL sont analysées, mais que l'objectif n'est pas trouvé, l'incident signalé est généré. Cet algorithme de recherche est généralement utilisé lorsque l'objectif se trouve dans les parties sans profondeur dans un arbre plus profond. [8]

Il utilise la frontière comme une file d'attente FIFO, rampant des liens dans l'ordre dans lequel ils sont rencontrés. Lorsque la frontière est pleine, le robot peut ajouter un seul lien.

L'algorithme est utilisé comme un crawler de ligne de base ; étant donné qu'il n'utilise aucune connaissance sur le sujet, nous nous attendons à ce qu'il fournisse une limite inférieure pour n'importe lequel des algorithmes les plus sophistiqués. [7]

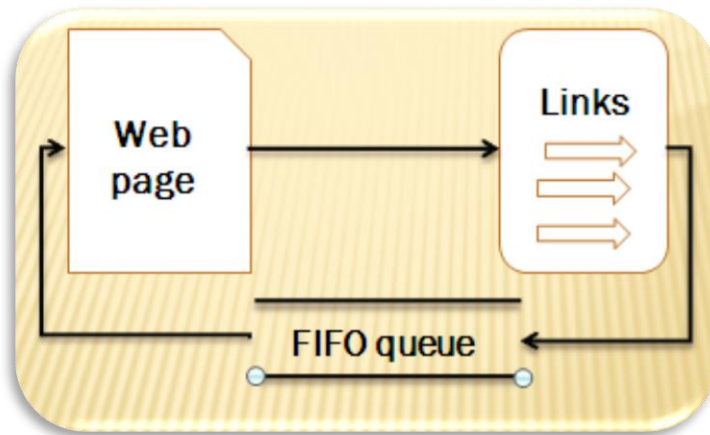


Figure 4: Parcours en largeur [7]

7.2. Algorithme de parcours en profondeur

Cette technique consiste à traverser en profondeur les différentes URL en commençant par l'URL jusqu'à l'URL enfant.

S'il y a plus d'un enfant, alors la priorité est accordée à l'enfant le plus à gauche et l'on continue à traverser en profondeur jusqu'à ce qu'il n'y ait plus d'enfants disponibles. Ensuite, la recherche est renvoyée au nœud non visité suivant, et se poursuit d'une manière similaire.

Cet algorithme assure que toutes les arêtes sont visitées au moins une fois en largeur. Il est bien adapté aux problèmes de recherche, mais lorsque les branches sont grandes, cet algorithme peut prendre fin dans une boucle infinie [8].



Figure 5. Parcours en profondeur [8]

7.3. Algorithme de Pagerank

Le PageRank d'une page représente la probabilité qu'un utilisateur aléatoire (celui qui suit des liens aléatoires d'une page à l'autre) soit sur cette page à un moment donné. Le score d'une page dépend récursivement des scores des pages qui pointent vers elle.

Les pages sources distribuent leur PageRank sur l'ensemble de leurs outlinks.

$$PR(p) = (1 - \gamma) + \gamma \sum_{\{d \in in(p)\}} \frac{PR(d)}{|out(d)|}$$

Où p est la page étant notée, dans (p) est l'ensemble de pages pointant vers p ,

Out (D) est l'ensemble des liens de d , et la constante $\gamma < 1$ est un facteur d'amortissement qui représente la probabilité que l'utilisateur aléatoire demande une autre page aléatoire.

Le PageRank initialement était destiné à être utilisé en combinaison avec des critères basés sur le contenu pour classer les ensembles de documents récupérés.

7.4.Algorithme génétique

L'algorithme génétique est basé sur l'évolution biologique par laquelle la progéniture la plus adaptée est obtenue en croisant de la sélection de quelques meilleurs individus dans la population au moyen de la fonction de conditionnement physique.

Dans un algorithme de recherche au problème existant, mais la technique consiste à trouver la meilleure solution dans le temps spécifié.

Montre que l'algorithme génétique est le mieux adapté lorsque l'utilisateur n'a littéralement pas ou pas moins de temps à consacrer à la recherche d'une base de données énorme et aussi très efficace dans les résultats multimédias. Alors, que presque toutes les méthodes conventionnelles de recherche à partir d'un seul point, Algorithmes génétiques fonctionne toujours sur une population entière. [9]

7.5.Algorithme de classification naïve bayésienne

L'algorithme de classification bayésienne est basé sur l'apprentissage probabiliste et la classification. Il suppose qu'une fonction est indépendante d'une autre.

Plusieurs auteurs ont proposé des crawlers efficaces basés sur la classification bayésienne afin de rassembler beaucoup de pages pertinentes pour des dispositions hiérarchiques de site web.

Cet algorithme s'est révélé efficace sur beaucoup d'autres approches, bien que son hypothèse simple ne soit pas très applicable dans des cas réalistes. [9]

7.6.Algorithme HITS

Cet algorithme proposé par Kleinberg est antérieur aux algorithmes de Page rank qui utilisent des scores pour calculer la pertinence. Cette méthode récupère un ensemble de résultats pour une recherche et calcule l'autorité et la note centrale dans cet ensemble de résultats. Cette méthode n'est pas souvent utilisée. [9]

II. Sécurité des services web

1. Définition

Un service web (en anglais web service) représente un mécanisme de communication entre applications distantes à travers le réseau Internet indépendant de tout langage de programmation et de toute plate-forme d'exécution

Les requêtes et les réponses s'effectuent dans des formats ouverts (HTML, XML, JSON ou text) et transitent par Internet.

2. La sécurité des services web

La sécurité des services web est très importante, puisqu'un service web est aussi perméable aux attaques et pénétrations que n'importe quel site web.

Pour cela, il y'a plusieurs méthodes et techniques pour atteindre des objectifs de sécurité.

3. Besoins de sécurité des services web

3.1. Autorisation

3.1.1. Types

- 1) **Par rôle.** (Contrôle d'accès basé sur des rôles) est une méthode d'autorisation dans laquelle les autorisations utilisateur sont gérées et sont appliquées par une application basée sur les rôles d'utilisateurs. Si un utilisateur a un rôle requis pour exécuter une action, l'accès est autorisé ; sinon, il est refusé.
- 2) **Par identité.** L'identité est de prendre une clé publique comme l'identité de l'utilisateur, par exemple, le nom, la date de naissance ou le numéro de sécurité sociale. Si l'on arrive à créer des clés de chiffrement secrètes relatives à ces identités de telle sorte que deux individus différents ne puissent avoir la même clé secrète, puis ne sont plus utiles pour l'authentification des clés publiques.
- 3) **Par ressources.** Lors de la création d'une page à partir d'une URL, le serveur HTTP utilise une ressource web principale (File://...) ainsi que des ressources complémentaires (AWEB_...), se cantonne généralement à garantir les droits d'accès aux données et ressources d'un système en mettant en place des mécanismes d'authentification et de contrôle permettant d'assurer que les utilisateurs des ressources possèdent uniquement les droits qui leur ont été octroyés.

3.1.2. Politique d'authentification

Mot de passe

Un mot de passe, ou password, est une séquence de caractères ou mot qu'un sujet donne à un système pour authentification, validation ou vérification. Un mot de passe fort permet davantage de sécurité.

Kerberos

Un jeton Kerberos comprend une authentification binaire et un jeton de session. Lorsqu'un jeton kerberos est utilisé avec une stratégie d'authentification uniquement, aucune clé privée n'est utilisée. Lorsqu'il est utilisé dans une stratégie qui inclut l'authentification et la protection des messages, les clés nécessaires à la protection des messages sont requises. [11]

Certificat X.509

Les messages de demande sont signés avec la clé de signature de l'utilisateur final. En ce qui concerne le client, on doit configurer un alias de clé de signature avec la clé de signature de l'utilisateur final. [11]

SAML

Le langage de balisage d'assertion de sécurité (SAML) est un cadre ouvert pour le partage d'informations de sécurité sur Internet via des documents XML.

Sécurité des services web

La sécurité des services web (WS-Security) spécifie les extensions de sécurité SOAP qui fournissent la confidentialité à l'aide du cryptage XML et de l'intégrité des données à l'aide de la signature XML. La sécurité des services web comprend également des profils qui spécifient comment insérer différents types de jetons de sécurité binaires et XML dans les en-têtes de sécurité web Services aux fins d'authentification et d'autorisation. [11]

3.2. Confidentialité

3.2.1. Chiffrement de flux

Le chiffrement de flux ou chiffrement par flot (en anglais stream cipher) est une des deux grandes catégories de chiffrements modernes en cryptographie symétrique, l'autre étant le chiffrement par bloc.

Un chiffrement par flot arrive à traiter les données de longueur quelconque et n'a pas besoin de les découper.

3.2.2. Chiffrement des données

Chiffrer des données (messages, images, etc.), cela consiste à les rendre illisibles à sauf si une action spécifique est exercée pour autoriser l'accès. La clé de voûte dans la protection des données aujourd'hui repose sur l'ensemble des techniques de chiffrement de données, que l'on appelle aussi techniques de cryptographie.

Le chiffrement est de transformer une donnée qui peut être lue par n'importe qui (donnée dite "claire") en une donnée qui ne peut être lue que par son créateur et son destinataire (donnée dite "chiffrée" ou encore cryptogramme). L'opération qui permet de récupérer la donnée claire à partir de la donnée chiffrée s'appelle le déchiffrement.

3.3. Authentification

3.3.1. Authentification par « digest »

L'authentification Digest est un mécanisme d'authentification dans lequel une application web s'authentifie auprès d'un service web en envoyant au serveur un résumé, qui est un hachage

cryptographique du mot de passe, nonce et timestamp. Lorsqu'on utilise l'authentification digest :

1- Le client effectue une demande non authentifiée au service web et le serveur envoie une réponse avec un test d'authentification digest indiquant qu'il prend en charge l'authentification Digest.

2- Le client génère un nonce et l'envoie au service avec un timestamp, digest et nom d'utilisateur. Le digest est un hachage cryptographique du mot de passe, nonce et timestamp.

3- Le serveur génère le hash lui-même à partir du mot de passe (récupéré dans le magasin de services), du nonce et du timestamp (à partir du message), et si le hash généré correspond au hash dans la requête, la requête est autorisée. [11]

3.4. Intégrité

3.4.1. Signature

Modèle de signature électronique (appelée aussi signature numérique) est un processus pour assurer la validité de l'expéditeur (fonction d'authentification) et de vérifier l'intégrité du message reçu. La signature électronique fournit également la fonction de non-répudiation, c'est-à-dire, il veille à ce que l'expéditeur ait bien envoyé le message autrement dit elle empêche l'expéditeur de nier avoir expédié le message).

4. Protection des données (confidentialité et Authentification)

4.1. Chiffrement à clé publique et privée

Le chiffrement à clé publique et privée est une méthode de chiffrement qui s'oppose à la cryptographie symétrique. Le principe du chiffrement asymétrique est d'avoir 2 clés (que l'utilisateur fabrique lui-même).

Par convention, on appelle une des 2 clés la clé privée et l'autre la clé publique.

La clé qui est choisie privée n'est jamais transmise à personne alors que la clé qui est choisie publique est transmissible sans restriction. Ce système permet deux choses majeures :

- Chiffrement du message à envoyer, l'expéditeur utilise la clé publique du destinataire pour chiffrer le message. Le destinataire utilise sa clé privée pour décoder le message de l'expéditeur, et d'assurer la confidentialité du contenu.
- S'assurer de l'authenticité de l'expéditeur, l'expéditeur utilise sa clé privée pour chiffrer le message que le destinataire peut décoder la clé publique de l'expéditeur. C'est le mécanisme utilisé par la signature numérique pour authentifier l'auteur d'un message.

4.2. Certificats

Le certificat est un document électronique utilisé pour identifier un individu, un serveur, une société ou une autre entité, et en liant la clé publique pour cette identité. Le certificat apporte la preuve de l'identité d'une personne généralement reconnue. Le chiffrement à clé publique utilise des certificats pour éviter les problèmes de vol d'identité.

Les autorités de certification (AC ou CA) sont des entités qui valident les identités et émettent les certificats. Le certificat émis par l'AC lie une clé publique particulière au nom de l'entité qu'il identifie (tel qu'un nom d'employé ou de serveur).

Les certificats aident à prévenir l'utilisation de fausses clés publiques pour l'usurpation d'identité. Seule la clé publique certifiée dans le certificat fonctionnera avec la clé privée correspondante possédée par l'entité identifiée par le certificat.

Le certificat contient toujours le nom de l'entité qui identifie, date d'expiration, le nom de son AC émettrice, numéro de série et toute autre information utile. Plus important encore, un certificat comprend toujours la signature numérique de l'AC émettrice.

La signature numérique de l'AC émettrice permet au certificat de fonctionner comme une lettre d'introduction pour les utilisateurs qui connaissent le secteur et lui font confiance, mais qui ne connaissent pas l'entité identifiée par le certificat.

4.3. SSL

SSL signifie Socket Secure Layer (littéralement Couche de Sécurité des Sockets, où les sockets sont les structures de communications sur la couche transport du modèle OSI). La version 3.1 du protocole a été renommée en TLS pour Transport Layer Security (Sécurité de la Couche Transport). [12]

4.3.1. Confidentialité

Assurer la confidentialité des données consiste à assurer que les données ne sont accessibles que par les personnes ayant un droit moral d'accès à ces données.

Dans le contexte de la communication entre les deux parties, et les données en question est l'information envoyée d'une partie à une autre, et la personne ayant le droit d'accéder aux données est le destinataire du message envoyé.

La confidentialité peut être compromise par un pirate tentant d'intercepter un message en transit sur le réseau et extraire des informations utiles. [12]

4.3.2. L'authentification

L'authentification consiste à s'assurer qu'une personne dit la vérité sur un ou plusieurs de ses attributs. Dans le contexte de la communication entre les deux parties, il peut être une caractéristique du parti ont leur son nom, sa raison sociale, ou tout autre élément de celui-ci est qu'ils savent, qui pourrait assurer l'autre partie de l'identité de sa partenaire.

Lors de l'accès à un site web de vente en ligne, le consommateur voudrait, au moment du paiement, être sûr qu'il donne son argent à la boutique en ligne dont il connaît la réputation, et non à un escroc qui encaissera l'argent sans suite. Voilà pourquoi il a besoin d'authentification : pour vous assurer que le site est la célèbre boutique en question, d'après son nom. Ainsi, il nécessite souvent l'identification du serveur web, mais rarement le client.

SSL permet l'authentification des deux parties. [12]

5. Autres besoins de sécurité à considérer

5.1. Traçabilité et audit

1) audit : effectuer un audit pour :

-Simuler le comportement d'un individu malveillant n'ayant aucune connaissance préalable des systèmes d'information.

-Identifier et à exploiter des failles de sécurité dans un ou plusieurs composants du système d'information.

-Identifier les faiblesses qui peuvent être exploitées par cette méthode détermine le succès du système vulnérable aux intrusions.

-Rapport sur les opérations, qui découle de l'analyse des résultats et permet d'améliorer la politique de sécurité que le contrôleur de système de test.

2) Traçabilité

La traçabilité est un système intégré dans la structure de production (atelier, site, usine, groupe ...), qui va permettre de suivre l'entité choisie à l'avance pour contrôler une autre, et le risque parfois avant et la qualité de l'entité. La traçabilité est composée principalement :

- d'une organisation dans la structure (les mesures à mettre en œuvre lors de la production)

- système d'information permettant d'enregistrer les données nécessaires à la constitution de l'historique de l'entité par une base de données relationnelle et des formulaires de saisie de l'information.

5.2. Utilisation de protocoles standards

Il existe principalement trois protocoles standards pour mettre en œuvre des services web : XML-RPC, REST et SOAP.

1) XML-RPC : est un protocole simple utilisant XML pour effectuer des messages RPC. Les requêtes sont écrites en XML et envoyées via HTTP POST. Les requêtes sont intégrées dans le corps de la réponse HTTP. XML-RPC est indépendant de la plate-forme, ce qui lui permet de communiquer avec diverses applications. [13]

2) REST : (Representationnel State Transfer) est une architecture de services web d'échange basé également sur HTTP. REST est une manière de construire une application pour les systèmes distribués comme le World Wide Web. Il n'utilise que les verbes standards du HTTP : GET, POST, PUT, DELETE. [13]

3) SOAP : (protocole Simple Object Access) est un protocole de communication standard. Ceci est l'épine dorsale de l'interopérabilité des systèmes. SOAP est un protocole décrit dans XML et normalisé par le W3C. Il se présente comme une enveloppe pouvant être signée et peut contenir des données ou des pièces jointes. Il fonctionne sur le protocole HTTP et permet d'effectuer les appels vers les méthodes distantes. [13]

4) WSDL : (Web Services Description Language) est un langage de description standard. L'interface est présentée aux utilisateurs. Il montre comment utiliser le service Web et comment interagir avec lui. WSDL est basé sur XML, et peut décrire avec précision les

détails du service web tels que les protocoles, ports utilisés, les opérations pouvant être effectuées, les formes des messages d'entrée et de sortie et les exceptions pouvant être envoyée. [13]

5) UDDI : (Universal Description, Discovery and Integration) est un annuaire de services. Il fournit l'infrastructure de base à publier et découvrir des services web. UDDI permet aux fournisseurs de présenter leurs services web aux clients. L'information peut être contenue séparé en trois types :

1-les pages blanches qui incluent l'adresse, le contact et les identifiants relatifs au service web.

2-les pages jaunes qui identifient les secteurs d'affaires relatifs au service web .

3-les pages vertes qui donnent les informations techniques. [13]

6. Scanner de sécurité des applications web

Les scanners de sécurité des applications Web sont des outils automatisés pour tester des applications Web pour des problèmes de sécurité communs tels que le Cross-Site Scripting, l'Injection SQL, le Traversage d'annuaires, les configurations non sécurisées et les vulnérabilités d'exécution de commandes à distance. Ces outils analysent une application Web et localisent les vulnérabilités et les faiblesses des couches d'application, soit en manipulant des messages HTTP, soit en les inspectant pour détecter des attributs suspects.

Un grand nombre d'outils d'analyse d'applications web sont disponibles, à la fois commerciaux et open sources.

L'utilisation efficace de ces outils est un élément important d'une évaluation approfondie de la sécurité des applications web et des analyses de sécurité régulières sont nécessaires pour se conformer aux exigences de sécurité.

Le WASSEC (Web Application Security Scanner Assessment Criteria) [10] est un ensemble de directives pour évaluer les scanners d'application web sur leur capacité à tester efficacement les applications web et à identifier les vulnérabilités. Il couvre des domaines tels que l'exploration, l'analyse syntaxique, la gestion des sessions, les tests et les rapports.

Chapitre 2 : Conception et modélisation

Introduction

De nombreuses attaques se focalisent sur les serveurs Web car ils sont souvent mal configurés ou mal maintenus. Les attaques les plus récentes profitent des failles de sécurité des applications Web. La détection de ces failles d'une trouve alors tous ses justificatifs en ce sens qu'à défaut de prévenir et protéger de façon sûre contre les attaques, les détecter permet néanmoins de limiter les dégâts et de réagir à temps. Dans ce chapitre nous présentons un bref récapitulatif sur les techniques de protection pour les applications Web et nous nous focalisons principalement sur les scanners de vulnérabilités Web et l'utilisation des crawlers pour la détection. La 1ère partie décrit la solution proposée. Ensuite, nous abordons la spécification des différents besoins de sécurité pour notre application. Enfin, la dernière partie, aborde la modélisation statique et dynamique du système proposé.

1. Présentation de la solution proposée

Nous présentons dans cette section la solution proposée. Tout d'abords, nous résumerons brièvement les travaux connexes à notre travail, ensuite nous mentionnerons quelques critiques sur ces travaux. Enfin, nous décrirons la solution proposée.

1.1. Description de l'existant

Dans cette section, nous décrivons d'abord les principes des scanners de vulnérabilités et nous analysons les limites de ces outils en nous basant, d'une part, sur l'analyse des algorithmes mis en œuvre dans certains scanners disponibles en source libre, et d'autre part, sur des études expérimentales publiées dans la littérature.

1.1.1. Principe des outils de détection de vulnérabilités

Il existe une grande variété de vulnérabilités visant les applications Web. Toutefois certaines sont plus connues et plus dangereuses que d'autres. Plusieurs bases de données répertorient ces vulnérabilités avec des statistiques indiquant leur importance relative. La multiplication des vulnérabilités et des attaques sur des sites web sur Internet ont poussé de nombreuses organisations à poser un regard critique sur la qualité de la sécurité de leurs applications web. Ainsi, plusieurs communautés ont vu le jour, dans le but d'améliorer la sécurisation des applications web. Les travaux dans ce contexte se sont traduits aussi par la proposition de taxonomies et de classifications pour les vulnérabilités et les attaques web les plus répandues. Parmi ces communautés, nous citons OWASP 4 (Open Web Application Security Project) et WASC5 (Web Application Security Consortium).

Les membres du "Web Application Security Consortium" ont créé ce projet pour développer et promouvoir une terminologie standard décrivant les problèmes de sécurité des applications Web et permettant aux développeurs d'applications, experts en sécurité, développeurs de logiciels et les consultants en sécurité, d'utiliser un langage commun pour interagir entre eux.

Les attaques les plus courantes concernant les serveurs Web sont les attaques d'injection SQL (lorsque le serveur Web est connecté à une base de données SQL) et d'injection de code

Javascript (réalisées sous la forme d'attaques de type Cross Site Scripting ou XSS). Ces injections de code proviennent de l'exploitation du même type de vulnérabilité des serveurs Web : l'absence de test de conformité des paramètres d'URL ou des données fournies dans les champs des formulaires.

Pour vérifier si ces attaques d'injection de code sont possibles, les outils de détection de vulnérabilités envoient des requêtes particulières et analysent les réponses retournées par le serveur. Un serveur peut répondre avec une page de rejet ou une page d'exécution. La page de rejet correspond à la détection par le serveur de valeurs d'entrée malformées ou invalides. Une page d'exécution est renvoyée par le serveur suite à l'activation réussie de la requête. Elle peut correspondre soit au scénario "normal", dans le cas d'une utilisation légitime du site, soit à un détournement de son exécution via l'exploitation réussie d'une injection de code (via des entrées non conformes).

Pour identifier les vulnérabilités d'un site Web, les outils de détection de vulnérabilités soumettent au site des requêtes contenant des données non conformes correspondant à des attaques potentielles. Les réponses sont alors analysées afin d'identifier les pages d'exécution. Si une page d'exécution est identifiée, la page correspondante est considérée vulnérable. C'est ainsi que les outils détectent l'absence de test de conformité des paramètres. Tout le problème vient donc de l'analyse des réponses pour déterminer s'il s'agit réellement d'une page de rejet ou d'une page d'exécution.

Prenons l'exemple d'une page d'authentification qui utilise une base de données SQL pour conserver les couples nom d'utilisateur / mot de passe valides. Un outil de détection de vulnérabilités doit déterminer si la page d'authentification est vulnérable à une injection SQL (une telle injection permettant à un attaquant de contourner l'authentification). A la requête d'authentification soumise, le serveur peut retourner deux types de réponses : succès ou échec de l'authentification. Ces deux catégories de réponses se traduisent généralement par l'affichage de deux pages différentes au niveau du navigateur du client en terme de code HTML de façon à ce que l'utilisateur puisse constater qu'il a entré un couple valide ou pas. Ces pages peuvent varier dans leur forme, en fonction du langage utilisé, du site lui-même, du développeur, etc. Les outils de détection de vulnérabilités doivent donc automatiquement classifier la réponse retournée afin de déterminer de façon correcte si la vulnérabilité est présente ou pas.

On peut distinguer deux principales classes d'approches adoptées par les scanners de vulnérabilités :

1) par reconnaissance de message d'erreurs dans les requêtes renvoyées par le serveur

Pour identifier une vulnérabilité de type « injections SQL » par exemple, cette approche consiste à envoyer des requêtes d'un format particulier et chercher des motifs spécifiques dans les réponses tels que les messages d'erreurs de base de données. L'idée fondamentale est que la présence d'un message d'erreur SQL dans une page HTML de réponse signifie que la requête correspondante n'a pas été vérifiée par l'application Web avant d'être transmise au serveur de bases de données. Par conséquent, le fait que cette requête a été envoyée inchangée

au serveur SQL révèle la présence d'une vulnérabilité. Les scanners tels que W3af¹ (module SQLI), Wapiti² et Secubat³ adoptent une telle approche.

2) *par l'étude de similarité des pages renvoyées.*

Le principe de cette approche consiste à envoyer différentes requêtes spécifiques aux types de vulnérabilités recherchées et à étudier la similitude des réponses renvoyées par l'application en utilisant une distance textuelle. En fonction des résultats obtenus et de critères bien définis, on conclut sur l'existence ou non d'une vulnérabilité. On peut citer comme exemple l'approche adoptée par Skipfish⁴ pour détecter les vulnérabilités d'injection SQL.

1.1.2. Analyse critique des scanners de vulnérabilités web

Les deux approches que nous venons de présenter présentent un certain nombre de limites.

L'efficacité de l'approche par reconnaissance de messages d'erreurs est liée à la complétude de la base de connaissance regroupant les messages d'erreurs susceptibles de résulter de l'exécution des requêtes soumises à l'application Web. Généralement, comme c'est dans le cas de W3af, on considère principalement les messages d'erreurs issus de la base de données. Cependant, les messages d'erreurs qui sont inclus dans des pages HTML de réponse ne proviennent pas forcément du serveur de bases de données. Le message d'erreur peut également être généré par l'application qui peut aussi reformuler le message d'erreur issu du serveur, par exemple pour le rendre compréhensible par le client. Par ailleurs, même si le message est généré par le serveur de base de données, la réception de ce message n'est pas suffisante pour affirmer que l'injection SQL est possible. En effet, ce message signifie que, pour cette requête particulière, les entrées n'ont pas été assainies, mais ne permet pas de conclure par rapport à d'autres requêtes SQL, en particulier celles qui seraient susceptibles de correspondre à des attaques réussies.

En ce qui concerne l'approche par similarité, elle se base sur l'hypothèse que le contenu d'une page de rejet est généralement différent du contenu d'une page d'exécution. Pour que cette comparaison puisse cependant être efficace, il est important d'assurer une large couverture des différents types de pages de rejet qui pourraient être générés par l'application. Ceci peut être réalisé en générant un grand nombre de requêtes visant à activer le plus grand nombre possible de pages de rejet variées. Cependant, les implémentations existantes de cette approche, en particulier dans Skipfish, génèrent trop peu de requêtes. Skipfish utilise seulement 3 requêtes. Si les réponses correspondent à différentes pages de rejet, il conclut à tort que la vulnérabilité est présente conduisant ainsi à un faux positif. Par ailleurs, pour l'approche par similarité, comme dans tout problème de classification, le choix de la distance est très important. Celle utilisée dans Skipfish ne prend pas en compte les ordres des mots dans un texte. Cependant, cet ordre définit généralement la sémantique de la page.

¹ <http://w3af.org/>

² <http://wapiti.sourceforge.net/>

³ <https://secubat.codeplex.com/>

⁴ <https://github.com/spinkham/skipfish>

1.2.Solution proposée

A l'issue de l'analyse des différents scanners de vulnérabilités, nous constatons qu'il y a matière à améliorer leurs performances. La solution décrite dans ce chapitre vise à atteindre cet objectif. En effet, nous proposons une approche permettant la détection automatisée des différents types de vulnérabilités Web, correspondant à différents types d'attaques qui seront décrit plus en détails dans la partie spécification des besoins.

Notre travail consiste à proposer un système de test de vulnérabilité de sécurité en utilisant un web crawler pour les applications web. La solution proposée permet de scanner les adresses URL pour tester du contenu Web malveillant et des vulnérabilités de sécurité communes. Ainsi, la solution que nous proposons est conçue pour répondre principalement à un problème spécifique, à savoir, la sécurité.

Pour traiter ce problème nous avons divisé notre travail en deux catégories : le crawl et le scan. La capacité de crawl est de trouver autant d'empreinte que possible au sein de l'application Web, tandis que le scan consiste à tester chaque partie des composants de l'application pour un ensemble appliqué de vulnérabilités.

A cet effet, nous proposons de développer une plateforme expérimentale intégrant le nouveau scanner de vulnérabilités, qui est destinée à évaluer l'efficacité des applications Web dans un contexte qui soit représentatif des menaces auxquelles ces applications seront confrontées en opération. Cette plateforme intègre plusieurs outils qui ont été conçus pour automatiser le plus possible les campagnes d'évaluation.

Nous nous sommes basée sur les techniques de crawling de pages Web, qui permet d'identifier les vulnérabilités à partir de l'analyse de cette page. Chaque vulnérabilité identifiée est réellement exploitée ce qui permet de s'assurer que la vulnérabilité identifiée ne correspond pas à un faux positif.

Une fois la crawling et le scanning effectué, nous obtenons une liste indexée des points de sécurité relatifs aux besoins de sécurité que nous allons détailler dans la partie suivante. L'évaluation finale se fait sur la base de cette liste. A partir de cette liste, nous procédons à la normalisation et comptabilisation des vulnérabilités obtenues et la valeur finale nous donnera la moyenne de ces différents points ce qui permettra d'interpréter le résultat final par rapport à l'application web comme étant très protégée, protégée, moyennement protégée, vulnérable et très vulnérable.

1.3.Choix du modèle proposé

L'approche proposée s'appuie sur certains concepts et outils existants et inclut des extensions significatives. Elle est basée sur la classification automatique des réponses retournées par les serveurs Web en utilisant les techniques de regroupement de données et permet d'identifier les vulnérabilités qui sont capables d'être exploiter. La génération automatique des requêtes permettant l'exploitation réussie des vulnérabilités est particulièrement utile pour faciliter la validation des applications Web.

Notre approche comporte deux étapes :

- 1) **Le Crawling** : La recherche du crawler d'empreintes et de points de vulnérabilités au sein de l'application Web
- 2) **Le Scanning** : la détection de vulnérabilités dans ces points, en testant chaque partie des points de l'application pour un ensemble appliqué de vulnérabilités.

Nous définissons un point de vulnérabilités par une entrée d'une page dans laquelle il est possible d'exploiter la vulnérabilité : un paramètre d'une URL ou un champ d'un formulaire.

La figure suivante présente une vue de haut niveau de la solution proposée. Elle commence par l'analyse de l'URL initiale (qui correspond, la plupart du temps, à la page principale de l'application). A partir de cette URL, l'exploration identifie tous les points potentiellement vulnérables. Cette première étape se termine lorsque tous les points accessibles ont été atteints. La deuxième étape correspond à l'exécution de notre algorithme de détection des vulnérabilités basée sur les besoins de sécurité (qui seront détaillés dans la 2ème partie du chapitre) sur chaque point. Cet algorithme permet l'identification et l'exploitation effective de vulnérabilités présentes. Il permet ainsi de découvrir de nouvelles pages qui peuvent contenir de nouveaux points qui n'étaient pas accessibles dans la première étape.

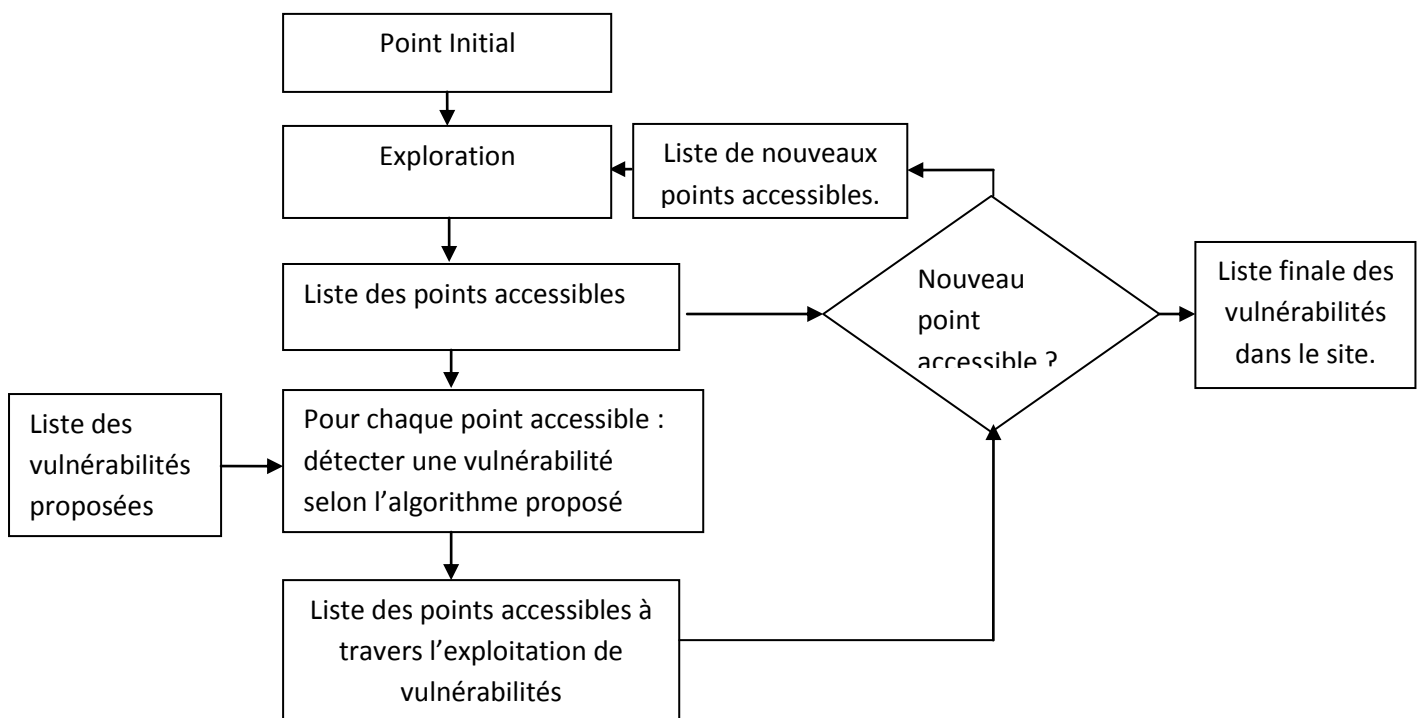


Figure 6. Extraction et recherche des points de vulnérabilités

1.3.1. Crawling

La première étape correspond au crawling du site permettant d'identifier les différentes navigations du site et ainsi les différents états de navigation. Notre approche est basée sur une recherche exhaustive permettant d'obtenir toutes les navigations possibles du site. Pour ce faire, nous prenons soin de partir d'un état initial. Cette initialisation est importante pour que les différentes navigations soient indépendantes. Ensuite, nous naviguons sur le site en commençant par le test initial et en mémorisant les tests envoyés au site. Le choix du test à envoyer se fait en analysant le contenu de la page affichée. Si cette page contient plusieurs liens HTML, un de ces liens est choisi pour construire le test suivant et les autres liens sont mémorisés pour les analyser ultérieurement. Sachant que la navigation à travers un site peut être infinie, nous bornons le nombre des tests envoyés. Cette borne représente la profondeur maximale de navigation du site. Lorsque nous aboutissons à une situation où la borne est atteinte ou l'état atteint ne permet plus d'envoyer des tests, autrement dit la page atteinte ne contient plus de liens HTML, alors la séquence de requêtes mémorisées constitue une navigation du site. Nous recommençons alors à zéro en essayant de nouvelles navigations, en se basant sur les choix mémorisés jusqu'à avoir tout essayé compte tenu de la borne fixée. Nous obtenons ainsi un ensemble de séquences de tests représentant l'ensemble de navigations du site. Cet ensemble est utilisé pour construire une première version du graphe de navigation dépourvu d'arcs correspondant à des vulnérabilités. L'objectif est d'obtenir un graphe minimal qui permet de représenter l'ensemble des navigations obtenues.

1.3.2. Scanning

La seconde étape vise à chercher les vulnérabilités qui peuvent être exploitées sur le site. Le graphe de navigation est utilisé pour effectuer cette recherche. Il est ensuite enrichi avec les vulnérabilités identifiées et utilisé lors des itérations suivantes. Etant donné le nombre important de requêtes envoyées pour obtenir ce graphe, mener une recherche de vulnérabilité sur chacune des requêtes envoyées serait excessivement long à réaliser. Toutes les requêtes du chemin choisi pour atteindre l'état doivent être exécutées dans l'ordre. Pour cela, nous choisissons le plus court chemin allant de l'état initial (avec des cookies vides) à cet état. Rappelons toutefois que l'algorithme est itératif. Il faut donc éviter de tester deux fois les arcs d'un même état. Cela revient à tester uniquement les arcs en sortie des nouveaux états, par rapport au graphe de l'itération précédente. Pour chacun des arcs, la recherche est réalisée sur tous les paramètres de la requête.

2. Spécification des besoins

Comme cité antérieurement, nous allons utiliser un web crawler afin de tester la sécurité d'une application web. Avant de procéder au test, il est primordial de définir les besoins de sécurité à garantir, pour cela nous nous sommes basés sur un des projets de la communauté OWSAP⁵ qui est OWASP Testing Guide.

⁵https://www.owasp.org/index.php/Main_Page

OWASP (Op/en Web Application Security Project) Open Web Application Security Project (OWASP) est une communauté en ligne travaillant sur la sécurité des applications Web. Sa philosophie est d'être à la fois libre et ouverte à tous. Elle a pour vocation de publier des recommandations de sécurisation Web et de proposer aux internautes, administrateurs et entreprises des méthodes et outils de référence permettant de contrôler le niveau de sécurisation de ses applications Web.

OWASP a proposé plusieurs projets parmi lesquels on retrouve OWASP Testing Guide⁶ qui représente un document de plusieurs centaines de pages destiné à aider une personne à évaluer le niveau de sécurité d'une application Web.

Notre travail a consisté à synthétiser ces différents tests de sécurité et d'essayer de les automatiser en se focalisant sur les besoins de sécurité qui sont pertinents par rapport à la solution proposée. Ainsi, les besoins qui nous semblent judicieux pour notre travail sont :

2.1. Collecte d'informations

2.1.1. Crawler le contenu effacé ou caché :

Les web Crawlers (ou Spiders) sont les outils les plus puissants et utiles utilisés pour de bonnes ou mauvaises intentions sur Internet. Une araignée sert une fonction majeure, l'exploration de données. La façon dont une araignée typique fonctionne est en rampant un site Web une page à la fois, la collecte et le stockage des informations pertinentes telles que les adresses e-mail, méta-tags, les données de formulaire masqué, les informations URL, les liens, Puis rampe tous les liens de cette page, collecte des informations pertinentes dans chaque page suivante, et ainsi de suite. A la fin le crawler peut récupérer des milliers de pages, rassemblant ainsi des morceaux d'information et stocke ces données dans une base de données.

L'objectif de ce test est de créer une carte de l'application avec tous les points d'accès (portes) de l'application et de récupérer toutes les informations publiées par l'application. Cela sera utile pour les tests de sécurité suivants.

2.1.2. Récupérer des métafichiers du serveur web pour des informations supplémentaires

Les web Spiders, Robots ou Crawlers récupèrent une page web et récursivement traversent des hypersliens pour récupérer d'autres contenus web. Leur comportement est spécifié par le protocole d'exclusion des robots (REP – Robots Exclusion Protocol) du fichier robots.txt. Le protocole d'exclusion des robots est une ressource de format texte qui peut être placée à la racine d'une application web, et qui contient une liste des ressources du site qui ne sont pas censées être indexées par les robots d'indexation des moteurs de recherche. Par convention, les robots consultent robots.txt avant d'indexer.

Il est important de souligner qu'il ne s'agit là que d'une indication sur ce que doivent faire les robots bienveillants, ce n'est en aucun cas un élément de sécurité. En effet, certains

⁶https://www.owasp.org/index.php/Category:OWASP_Testing_Project

robots ignorent ce fichier, soit délibérément parce qu'ils cherchent des informations privées (des adresses électroniques par exemple, pour y envoyer du courrier indésirable), soit parce que le robot est trop simple pour gérer ce standard.

2.1.3. Examiner les commentaires et les métadonnées des pages web

Il est très fréquent pour les programmeurs d'inclure des commentaires détaillés et des métadonnées sur leur code source. Toutefois, les commentaires et les métadonnées inclus dans le code HTML pourraient révéler des informations internes qui ne devraient pas être disponibles pour les éventuels attaquants. Les commentaires et la révision des métadonnées doivent être effectués afin de déterminer si une information est divulguée.

Les commentaires HTML sont souvent utilisés par les développeurs pour inclure des informations de débogage sur l'application. Parfois, ils oublient les commentaires et ils les laissent en production. Pour tester ce besoin il est primordial de vérifier le code source HTML pour les commentaires contenant des informations sensibles qui peuvent aider l'attaquant à mieux comprendre l'application. Il peut s'agir de code SQL, de noms d'utilisateur et de mots de passe, d'adresses IP internes ou d'informations de débogage. Exemple : `<!-- Use the DB administrator password for testing: f@keP@a$$w0rD -->`

2.1.4. Vérifier la plateforme de l'application web

Connaître le type de la plateforme peut donner un grand avantage si une telle plateforme a déjà été testée. En plus des vulnérabilités connues dans les versions non patchées, on peut aussi utiliser cette vérification pour prendre avantage de mauvaises configurations ou de la structure des fichiers de la plateforme.

Plusieurs éditeurs et versions de plateformes web sont utilisés. Les informations à ce sujet aident considérablement le processus de test. Ces informations peuvent être obtenues par une analyse minutieuse de certains emplacements. La plupart des plateformes web ont plusieurs marqueurs dans ces emplacements qui aident un attaquant à les repérer.

Pour définir le type de plateforme utilisée il faut comprendre la méthodologie de test de sécurité. De plus, il y'a plusieurs emplacements communs qu'il faut vérifier afin de définir la plateforme, on peut citer :

- En-têtes HTTP : la forme la plus simple pour identifier une plateforme web est de vérifier le champ « *X-Powered-By* » dans l'entête de réponse HTTP.
- Cookies : Un autre moyen semblable et en quelque sorte plus fiable pour déterminer la plateforme Web sont les cookies spécifiques à la plateforme.
- Code source HTML : Cette technique est basée sur la recherche de certains patterns dans le code source de la page HTML. Souvent, on peut trouver beaucoup d'informations qui aide à reconnaître une plateforme web spécifique. Un des marqueurs communs sont les commentaires HTML qui conduisent directement à la divulgation de la plateforme.
- Extensions de fichiers: L'URL peut inclure des extensions de fichier. Les extensions de fichier peuvent également aider à identifier la plate-forme Web ou la technologie.

Par exemple : .php pour PHP, .aspx pour Microsoft ASP.NET, .jsp pour Java Server pages.

- Fichiers et dossiers spécifiques.
- Message d'erreur.

2.1.5. Tester les définitions des rôles

Il est courant dans les entreprises de définir les rôles système pour gérer les utilisateurs et l'autorisation d'accès aux ressources système. Dans la plupart des implémentations du système, il est prévu qu'au moins deux rôles existent, les administrateurs et les utilisateurs réguliers. Le premier représente un rôle qui permet d'accéder à des fonctionnalités et des informations privilégiées et sensibles, le second représentant un rôle qui permet d'accéder à des fonctionnalités et des informations commerciales régulières. Les rôles bien développés doivent s'aligner sur les processus métier soutenus par l'application.

Pour tester il faut valider les rôles système définis dans l'application en définissent et séparent suffisamment chaque rôle pour gérer l'accès approprié aux fonctionnalités et aux informations du système.

2.1.6. Identifier les points d'entrée de l'application

Enumérer l'application et de sa surface d'attaques est un précurseur clé avant qu'un test approfondi puisse être entrepris car il permet d'identifier les zones de faiblesses et vulnérabilités probables.

Avant de commencer le test, il faut accorder une attention particulière à toutes les requêtes HTTP (méthodes GET et POST) ainsi qu'à tous les paramètres et champs de formulaire transmis à l'application.

Les points à identifier pour les requêtes :

- Identifier où les GET sont utilisés et où les POST sont utilisés.
- Identifier tous les paramètres utilisés dans une demande POST (ceux-ci sont dans le corps de la demande).
- Dans la requête POST, faire particulièrement attention aux paramètres cachés. Lorsqu'un POST est envoyé, tous les champs de formulaire (y compris les paramètres cachés) seront envoyés dans le corps du message HTTP à l'application. Ceux-ci ne sont généralement pas vu à moins qu'un proxy ou afficher le code source HTML est utilisé.
- Identifier tous les paramètres utilisés dans une requête GET (c.-à-d., URL), en particulier la chaîne de requête (généralement après '?').
- Identifier tous les paramètres de la chaîne de requête. Ceux-ci sont généralement dans un format de pair, tel que foo = bar. Notez également que de nombreux paramètres peuvent être dans une chaîne de requête telle que séparée par un &, ~, :, ou tout autre caractère spécial ou encodage.
- Faites également attention à tous les en-têtes de type supplémentaires ou personnalisés qui ne sont généralement pas vus (tels que debug = False).

Les points à identifier pour les réponses :

- Identifier l'emplacement des nouveaux cookies (en-tête Set-Cookie), leur modification ou ajout à.
- Identifier les redirections (code d'état 3xxHTTP), les codes d'état 400, en particulier 403 Interdit, et 500 erreurs de serveur interne pendant les réponses normales (c'est-à-dire les requêtes non modifiées).

2.2. Transmission Sécurisée :

2.2.1. Vérifier les protocoles SSL /TLS

Les données sensibles doivent être protégées lorsqu'elles sont transmises sur le réseau. Ces données peuvent inclure des informations d'identification d'utilisateur et des cartes de crédit. En règle générale, si les données doivent être protégées lorsqu'elles sont stockées, elles doivent également être protégées pendant la transmission. HTTP est un protocole en clair et il est normalement sécurisé via un tunnel SSL / TLS, ce qui entraîne un trafic HTTPS. L'utilisation de ce protocole garantit non seulement la confidentialité, mais aussi l'authentification. Les serveurs sont authentifiés à l'aide de certificats numériques et il est également possible d'utiliser le certificat client pour l'authentification mutuelle.

Même si des chiffrements de haut niveau sont aujourd'hui supportés et normalement utilisés, une mauvaise configuration du serveur peut être utilisée pour forcer l'utilisation d'un chiffrement faible - ou au pire pas de cryptage - permettant à un attaquant d'accéder au canal de communication sécurisé supposé. Une autre mauvaise configuration peut être utilisée pour une attaque de déni de service.

Pour identifier les problèmes de sécurité pour cette section :

- Une vulnérabilité se produit si le protocole HTTP est utilisé pour transmettre des informations sensibles (par exemple des informations d'identification transmises via HTTP).
- L'utilisation du service SSL / TLS est bien mais augmente la surface d'attaque et les vulnérabilités suivantes existent:
 - Les protocoles SSL / TLS, les chiffres, les clés et la renégociation doivent être correctement configurés.
 - La validité du certificat doit être assurée.
- On peut retrouver d'autres vulnérabilités telles que :
 - Les logiciels exposés doivent être mis à jour en raison de la possibilité de vulnérabilités connues.
 - Utilisation d'un drapeau sécurisé pour les cookies de session.
 - Utilisation de la sécurité de transport stricte HTTP (HSTS – HTTP Strict Transport Security).
 - La présence de HTTP et HTTPS à la fois, qui peuvent être utilisés pour intercepter le trafic.

- La présence de contenu mixte HTTPS et HTTP dans la même page, qui peut être utilisé pour récupérer des informations.

2.3. Authentification

2.3.1. Tester les règles de qualité des mots de passe

Le mécanisme d'authentification le plus répandu et le plus facilement administré est un mot de passe statique. Dans chacun des récents piratages de haut niveau qui ont révélé les informations d'identification utilisateur, il est déploré que les mots de passe les plus courants soient encore: 123456, password et qwerty.

Cette section consiste à déterminer la résistance de l'application contre la détection de mots de passe par force brute en utilisant des dictionnaires de mots de passe disponibles en évaluant la longueur, la complexité, la réutilisation et le vieillissement des mots de passe.

Pour limiter le risque de mots de passe faciles permettant l'accès non autorisé, il existe deux solutions : introduire des contrôles d'authentification supplémentaires (c'est-à-dire l'authentification à deux facteurs) ou introduire une stratégie de mots de passe forts. Le plus simple et le moins cher de ces derniers est l'introduction d'une politique de mot de passe fort qui assure la longueur de mot de passe, la complexité, la réutilisation et le vieillissement.

2.3.2. Tester de l'énumération des utilisateurs

La portée de ce test est de vérifier s'il est possible de collecter un ensemble de noms d'utilisateurs valides en interagissant avec le mécanisme d'authentification de l'application. Souvent, les applications Web révèlent quand un nom d'utilisateur existe sur le système, soit en conséquence d'une mauvaise configuration, soit en tant que décision de conception. Par exemple, parfois, lorsque nous envoyons des informations d'identification erronées, nous recevons un message indiquant que le nom d'utilisateur est présent sur le système ou que le mot de passe fourni est incorrect. Les informations obtenues peuvent être utilisées par un attaquant pour obtenir une liste d'utilisateurs sur le système. Ces informations peuvent être utilisées pour attaquer l'application Web, par exemple, par une attaque brutale ou une attaque par nom d'utilisateur / mot de passe par défaut.

On peut énumérer les utilisateurs de diverses façons, telles que :

- Analyser le code d'erreur reçu sur les pages d'authentification.
- Analyser les URLs et les redirections des URLs.
- Analyser les titres des pages web.
- Analyser les messages reçus lors de récupération (comme mot de passe oublié).
- Deviner les utilisateurs (USER01, USER02, ...).

2.3.3. Tester le contournement du schéma d'authentification

Bien que la plupart des applications requièrent une authentification pour accéder à des informations privées ou pour exécuter des tâches, toutes les méthodes d'authentification ne sont pas en mesure de fournir une sécurité adéquate. La négligence, l'ignorance ou la simple sous-évaluation des menaces de sécurité entraînent souvent des schémas d'authentification qui

peuvent être contournés en ignorant simplement la page de connexion et en appelant directement une page interne qui est censée être accessible uniquement après l'authentification a été effectuée.

En outre, il est souvent possible de contourner les mesures d'authentification en altérant les requêtes et en incitant l'application à penser que l'utilisateur est déjà authentifié. Cela peut être accompli soit en modifiant le paramètre d'URL donné, en manipulant le formulaire ou en falsifiant des sessions. On peut citer :

- Demande de page directe (navigation forcée)
- Modification des paramètres
- Prédiction de l'ID de session
- SQL injection

2.3.4. Tester la transmission d'informations d'authentification sur un canal crypté

Le test du transfert d'informations d'identification consiste à vérifier que les données d'authentification de l'utilisateur sont transférées via un canal crypté pour éviter d'être interceptées par des utilisateurs malveillants. L'analyse se concentre sur la vérification de la transmission des données du navigateur web vers le serveur et si elle est cryptée, ou si l'application Web prend les mesures de sécurité appropriées en utilisant un protocole comme HTTPS. Le protocole HTTPS est basé sur TLS / SSL pour chiffrer les données transmises et pour s'assurer que l'utilisateur est envoyé vers le site souhaité.

De toute évidence, le fait que le trafic est crypté ne signifie pas nécessairement qu'il est complètement sûr. La sécurité dépend également de l'algorithme de cryptage utilisé et de la robustesse des clés utilisées par l'application.

Le test de cette section consiste à vérifier la transmission des données avec POST et GET à travers HTTP et HTTPS.

2.3.5. Tester par force brute

Le test par force brute consiste à énumérer systématiquement tous les candidats possibles à la solution et à vérifier si chaque candidat satisfait à la déclaration du problème. Dans les tests d'application Web, le problème auquel nous faisons face le plus souvent est lié à la nécessité d'avoir un compte d'utilisateur valide pour accéder à la partie interne de l'application. Par conséquent, il faut vérifier les différents types de schéma d'authentification et l'efficacité des différentes attaques par force brute.

Pour tirer parti des différentes attaques par force brut, il est important de découvrir le type de méthode d'authentification utilisée par l'application, car les techniques et les outils à utiliser peuvent changer en conséquence. À moins qu'une entité décide d'appliquer une authentification Web sophistiquée, les deux méthodes les plus couramment vues sont les suivantes :

- Authentification http : Authentification d'accès basique, Authentification d'accès Digest

- Authentification HTML basée sur formulaire ;

Une fois la méthode d'authentification récupérée, on peut tester l'attaque par force brute. Pour cela il y'a plusieurs types d'attaque par force brute. On peut citer :

- Attaque par dictionnaire
- Attaque par recherche
- Attaque par recherche basée sur les règles.

2.4. Cryptographie

2.4.1. Vérifier si les données à crypter ne le sont pas

Les données sensibles doivent être protégées lorsqu'elles sont transmises par le réseau. Si les données sont transmises via HTTPS ou chiffrées d'une autre manière, le mécanisme de protection ne doit pas présenter de limitations ou de vulnérabilités. Voici quelques exemples de données sensibles:

- Informations utilisées lors de l'authentification (p. Ex., Informations d'identification, NIP, identifiants de session, jetons, cookies ...)
- Informations protégées par des lois, des règlements ou une politique organisationnelle spécifique (par exemple Cartes de crédit, Données clients)

De nombreux types d'informations qui doivent être protégés, pourraient être transmis par l'application en texte clair. Il est possible de vérifier si ces informations sont transmises via HTTP au lieu de HTTPS, ou si des chiffreurs faibles sont utilisés. Pour vérifier cette section, on peut citer :

- Authentification basique via HTTP.
- Authentification basée formulaire via http
- Cookies contenant l'ID de Session via http

2.4.2. Vérifier l'utilisation d'algorithmes faibles

De la même façon, la vérification de la transmission des données sensibles à travers le réseau ainsi que la vérification des algorithmes utilisés. Pour cela :

- Tester l'utilisation d'un algorithme ou si les données sont transmises en clair
- Les chiffreurs faibles ne doivent pas être utilisés (par exemple, moins de 128 bits, aucune suite de codage NULL aucune Diffie-Hellmann anonyme).
- Les protocoles faibles doivent être désactivés (par exemple SSLv2 doit être désactivé, en raison de faiblesses connues dans la conception du protocole).
- La renégociation doit être correctement configurée (par exemple, la renégociation INSECURE doit être désactivée en raison des attaques MiTM et la renégociation initiée par le client doit être désactivée en raison de la vulnérabilité de déni de service).
- La longueur des clés X.509 doit être forte (par exemple si RSA ou DSA est utilisée, la clé doit être au moins 1024 bits).

- Les certificats X.509 doivent être signés uniquement avec des algorithmes de hachage sécurisés (Ex. Non signés à l'aide du hachage MD5, en raison d'attaques de collision connues sur ce hachage).

2.5. Validation des données

2.5.1. Test pour injection SQL

Une attaque par injection SQL consiste à insérer ou "injecter" une requête SQL partielle ou complète via la saisie de données ou transmise par le client (navigateur) à l'application Web. Une attaque d'injection SQL réussie peut lire des données sensibles à partir de la base de données, modifier les données de la base de données (insérer / mettre à jour / supprimer), exécuter les opérations d'administration sur la base de données (par exemple, arrêter le SGBD), récupérer le contenu d'un fichier donné existant sur le système du SGBD ou écrire des fichiers dans le système de fichiers et, dans certains cas, émettre des commandes sur le système d'exploitation.

Les attaques d'injection SQL peuvent être divisées en trois classes :

- 1) In-band: les données sont extraites à l'aide du même canal utilisé pour injecter le code SQL. C'est l'attaque la plus directe, dans laquelle les données récupérées sont présentées directement dans la page Web de l'application.
- 2) Out-of-band: les données sont récupérées à l'aide d'un canal différent (par exemple, un email avec les résultats de la requête est généré et envoyé au testeur).
- 3) Inférentiel ou Blind: il n'y a pas de transfert réel de données, mais le testeur est capable de reconstruire l'information en envoyant des requêtes particulières et en observant le comportement résultant du serveur DB.

Il existe principalement cinq techniques communes pour exploiter les vulnérabilités d'injection SQL. De plus, ces techniques peuvent parfois être utilisées de manière combinée (par exemple, opérateur union et hors bande):

- Opérateur d'Union : peut être utilisé lorsque l'injection SQL se produit dans une instruction SELECT, permettant de combiner deux requêtes en un seul résultat ou ensemble de résultats.
- Booléenne : utilise les conditions booléennes pour vérifier si certaines conditions sont vraies ou fausses.
- Basée Erreur : cette technique force la base de données à générer une erreur, en donnant aux informations de l'attaquant sur lesquelles affiner leur injection.
- Hors Bande : technique utilisée pour récupérer des données à l'aide d'un canal différent (par exemple, faire une connexion HTTP pour envoyer les résultats à un serveur Web).
- Temporisation: utilise les commandes de la base de données (par exemple, sleep) pour retarder les réponses dans les requêtes conditionnelles. Cette technique est utile lorsque l'attaquant n'a pas de réponse (résultat, sortie ou erreur) de l'application.

2.5.2. Test pour XSS (Cross Site Scripting) réfléchi

Le XSS réfléchi se produit lorsqu'un attaquant injecte un code exécutable du navigateur dans une seule réponse HTTP. L'attaque injectée n'est pas stockée dans l'application elle-même; elle est non persistante et n'affecte que les utilisateurs qui ouvrent un lien malin ou un site Web tiers. La chaîne d'attaque est incluse dans les paramètres URI ou HTTP, mal traitée par l'application et retournée à la victime. Le XSS réfléchi est le type d'attaque XSS le plus fréquent trouvé dans la nature. Les attaques XSS reflétées sont également connues sous le nom d'attaques XSS non persistantes ou XSS de premier ordre ou de type 1.

Généralement, le code de l'attaquant est écrit dans la langue Javascript, mais d'autres langues de script sont également utilisées, par exemple, ActionScript et VBScript. Les attaquants utilisent généralement ces vulnérabilités pour installer les enregistreurs de clés, voler les cookies des victimes, effectuer un vol de presse papier et modifier le contenu de la page (par exemple, télécharger des liens).

Le test de ce type d'attaque comprend au moins trois phases:

- 1) Détecter les vecteurs d'entrée. Pour chaque page Web, le testeur doit déterminer toutes les variables définies par l'utilisateur de l'application Web et comment les saisir. Cela comprend des entrées cachées ou non évidentes telles que les paramètres HTTP, les données POST, les valeurs du champ de formulaire caché et les valeurs de radio ou de sélection prédéfinies. Typiquement, les éditeurs HTML dans le navigateur ou les proxys Web sont utilisés pour afficher ces variables cachées.
- 2) Analyser chaque vecteur d'entrée pour détecter les vulnérabilités potentielles. Pour détecter une vulnérabilité XSS, le testeur utilisera généralement des données d'entrée spécialement conçues avec chaque vecteur d'entrée. Ces données d'entrée sont généralement inoffensives, mais déclenchent des réponses du navigateur Web qui manifeste la vulnérabilité.
- 3) Pour chaque entrée de test tentée dans la phase précédente, le testeur analysera le résultat et déterminera s'il représente une vulnérabilité qui a un impact réaliste sur la sécurité de l'application Web. Cela nécessite d'examiner la page Web HTML résultante et la recherche de l'entrée de test. Une fois trouvé, le testeur identifie les caractères spéciaux qui n'ont pas été correctement codés, remplacés ou filtrés. L'ensemble des caractères spéciaux non filtrés vulnérables dépendra du contexte de cette section de HTML.

3. Conception

Dans cette partie, nous allons présenter la modélisation de l'application d'un point de vue statique et dynamique.

3.1. Langage UML

Les techniques de programmation n'ont cessé de progresser depuis l'époque de la programmation en langage binaire à nos jours. Cette évolution a toujours été dictée par le besoin de concevoir et de maintenir des applications toujours plus complexes.

La technologie objet est donc la conséquence ultime de la modélisation dictée par la maîtrise de la conception et de la maintenance d'applications toujours plus complexes. Cette nouvelle technique de programmation a nécessité la conception de nouvelles méthodes de modélisation.

Notre travail consiste à réaliser un web crawler pour assurer la sécurité pour les applications Web. Et afin de faire une bonne réalisation, il faut faire une bonne modélisation. Dans notre travail, pour l'étape de modélisation, nous avons choisi la méthode de modélisation UML.

Notre démarche de travail consiste en premier lieu en la création des diagrammes de cas utilisation, de séquences et du diagramme de collaboration et de classe, puis nous y associons un diagramme d'activités. Et enfin, nous terminerons par l'architecture de notre application.

3.1.1. Définition UML

UML «Unified Modeling Language», est langage visuel qui se compose d'un ensemble de schémas appelés diagrammes qui donnent une vision différente du projet à traiter. Ainsi, UML fournit des diagrammes pour représenter le programme lors de son fonctionnement, de sa mise en route et les actions susceptibles d'être effectuées par le logiciel.

3.1.2. Les diagrammes UML

UML 2.0 comporte treize types de diagrammes représentant autant de *vues* distinctes pour représenter des concepts particuliers du système d'information. Ils se répartissent en deux grands groupes : diagrammes statiques et diagrammes dynamiques. [23]

3.1.2.1. Diagrammes structurels ou diagrammes statiques (*UML Structure*)

- Diagramme de classes (*Class diagram*). Il représente les classes intervenant dans le système. Le diagramme de classe est une représentation statique des éléments qui composent un système et de leurs relations.
- Diagramme d'objets (*Object diagram*). Il sert à représenter les instances de classes (objets) utilisées dans le système.
- Diagramme de composants (*Component diagram*). Il permet de montrer les composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données...)
- Diagramme de déploiement (*Deployment diagram*). Il sert à représenter les éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage...) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent avec eux.
- Diagramme de paquetages (*Package diagram*). Un paquetage étant un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML, le Diagramme de paquetage sert à représenter les dépendances entre paquetages, c'est-à-dire les dépendances entre ensembles de définitions.

- Diagramme de structures composites (*Composite structure diagram*). Permet de décrire sous forme de boîte blanche les relations entre composants d'une classe. [24]

3.1.2.2. Diagrammes comportementaux ou diagrammes dynamiques (*UML Behavior*)

- Diagramme de cas d'utilisation (*Use case diagram*). Il permet d'identifier les possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire toutes les fonctionnalités que doit fournir le système. Il permet aussi de délimiter le système.

- Diagramme d'activités (*Activity diagram*). Il permet de décrire sous forme de flux ou d'enchaînement d'activités le comportement du système ou de ses composants.

- Diagramme d'états-transitions (*State machine diagram*). Il permet de décrire sous forme de machine à états finis le comportement du système ou de ses composants.

- Diagrammes d'interaction (*Interaction diagram*).

- Diagramme de séquence (*Sequence diagram*). Représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.

- Diagramme de communication (*Communication diagram*). Représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.

- Diagramme global d'interaction (*Interaction overview diagram*). Permet de décrire les enchaînements possibles entre les scénarii préalablement identifiés sous forme de diagrammes de séquence (variante du diagramme d'activité).

- Diagramme de temps (*Timing diagram*). Permet de décrire les variations d'une donnée au cours du temps. [25]

3.2. Modélisation dynamique

Dans ce qui suit, nous présentons la modélisation dynamique de notre application.

3.2.1. Diagramme de séquence

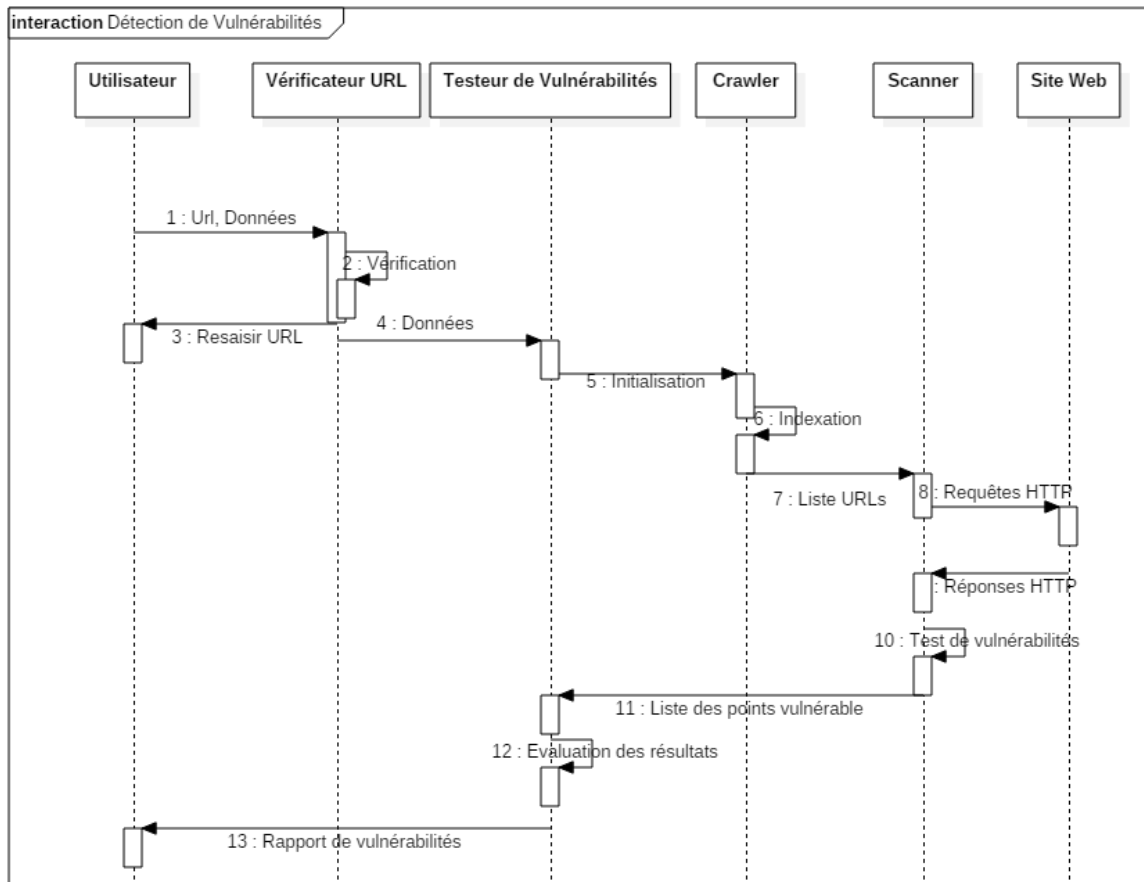


Figure 7. Diagramme de séquence « Détection de vulnérabilités »

Scénario:

Quand l'utilisateur envoie l'URL de l'application web à tester, il y'a d'abord vérification que l'URL est correcte, ensuite l'application lance le crawler afin de pouvoir indexer toutes les entrées possibles à tester. Une fois cette étape terminée le crawler envoie la liste des URLs au scanner afin de pouvoir lancer le test de vulnérabilité par rapport à chaque entrée. Ensuite, le scanner renvoie les entrées vulnérables à l'application afin de procéder à l'évaluation du résultat final. Enfin, un rapport final est généré pour l'utilisateur.

3.2.2. Diagramme de collaboration :

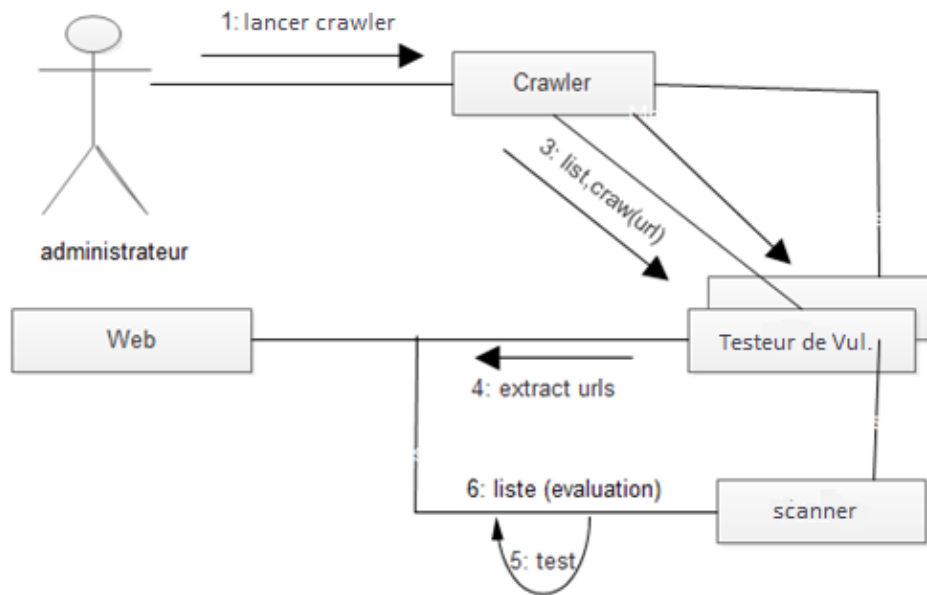


Figure 8. Diagramme de collaboration «Test de vulnérabilités»

3.2.3. Diagrammes d'activités

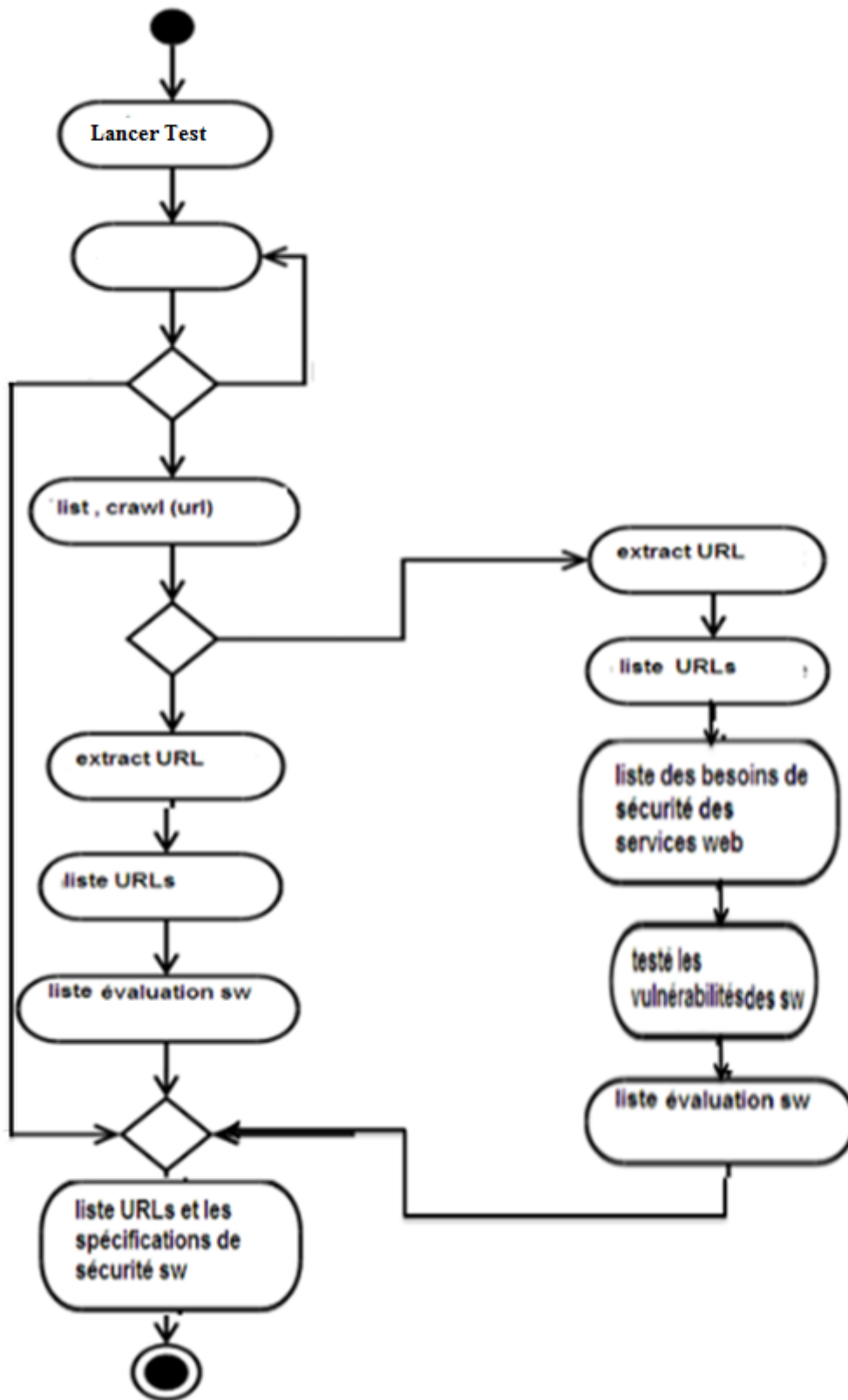


Figure 9. Diagramme d'activités «Evaluation de vulnérabilités».

3.3. Modélisation statique

3.3.1. Diagrammes de classes

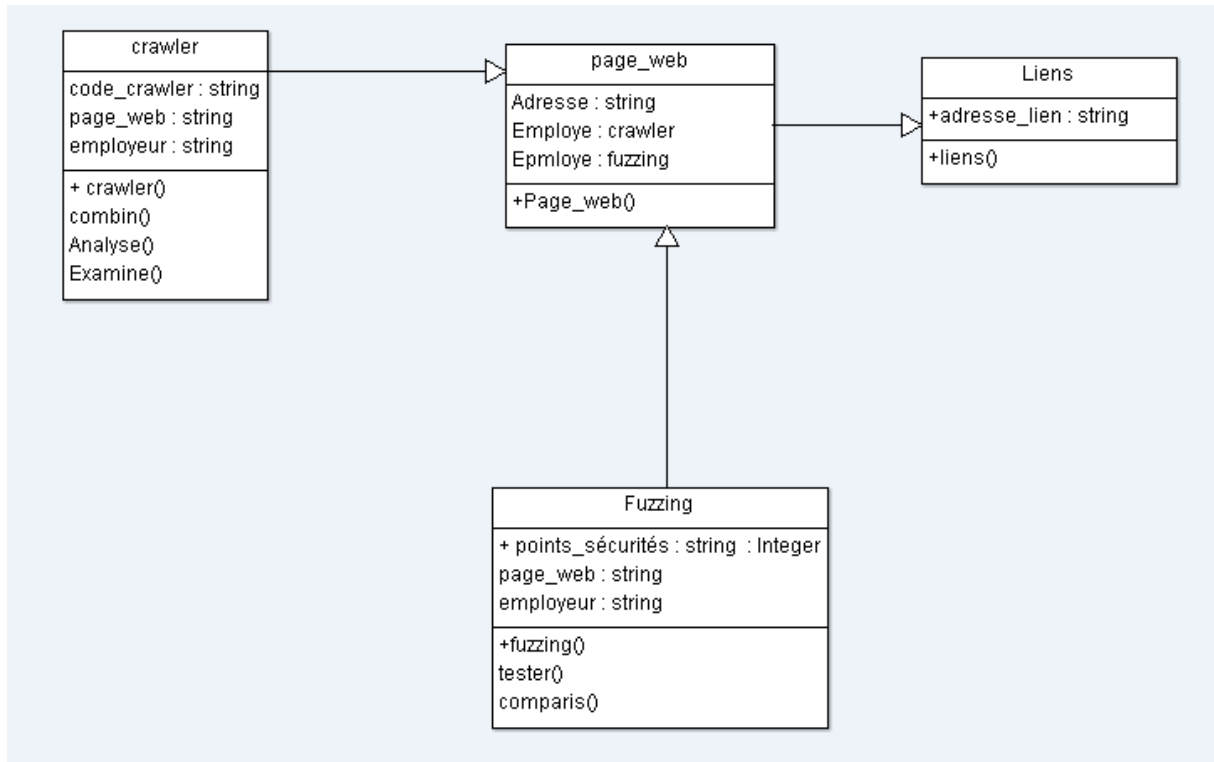


Figure : Diagrammes de classes «Crawler»

3.3.2 Architecture de l'application

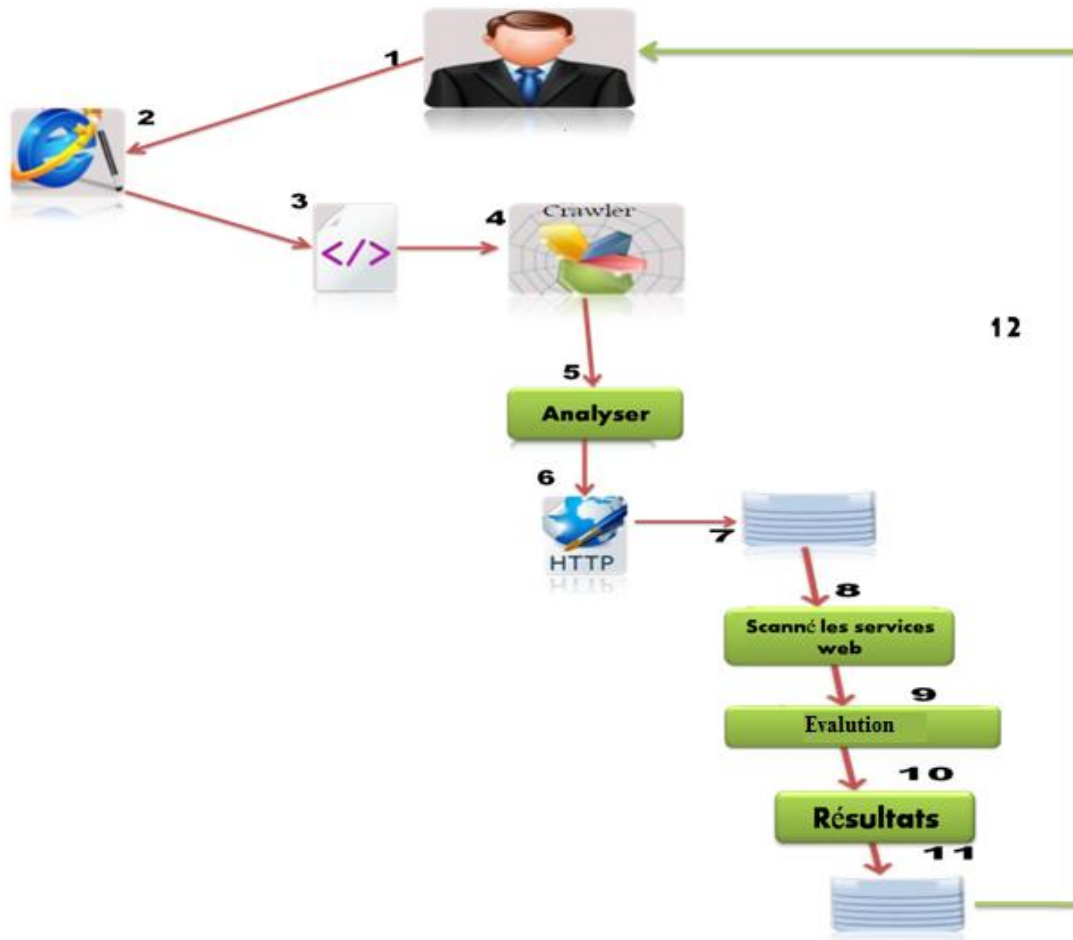


Figure 10. Architecture générale de notre application

Explication de l'architecture

- (1) l'utilisateur entre une requête.
- (2) Application obtient la requête.
- (3) envoi liste d'url d'index
- (4) crawler recevoir l'ordre de démarrage d'analyse
- (5) extraction liens
- (6) identification liens
- (7) stock les données à base d'indexation
- (8) le scanner test la liste des URLs avec les besoins de sécurité
- (9) évaluation des applications web
- (10) afficher les résultats
- (11) sauvegarder dans la base de donner les résultats finaux
- (12) Envoyer la réponse à l'utilisateur

Conclusion

Dans ce chapitre, nous avons présenté un scanner de vulnérabilité qui permet d'identifier la présence de certains types de vulnérabilités et fournit les requêtes permettant d'exploiter ces vulnérabilités.

Dans le but de faciliter l'implémentation de notre outil de recherche, dans ce chapitre, nous avons aussi présenté notre modélisation par des digrammes en UML afin de cerner les besoins de sécurité. Les résultats de ce chapitre seront enrichis par des détails d'implémentation dans le chapitre suivant pour la réalisation de l'application.

Chapitre 3 :

Mise en œuvre et

implémentation

Introduction

Dans ce chapitre, nous présentons l'implémentation de l'application conçue pour le crawling et la détection des vulnérabilités des applications web.

Ce chapitre est essentiellement consacré aux langages et outils exploités pour créer les différentes parties de ce travail. Des captures d'écran seront aussi présentées.

1. Environnement du travail

1.1. Environnement Matériel

La configuration matérielle qu'on a utilisé est la suivante :

- Ordinateur Acer
- Système d'exploitation : Windows 7 - 64 bites
- Processeur : Intel(R) pentium (R) CPU N3520@ 2.16Ghz
- Mémoire : 2 Go

1.2. Environnement logiciel

1.2.1. Python

Python est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles.

Pourquoi python ?

Python offre vraiment un niveau d'apprentissage rapide. Grâce à la simplicité de sa syntaxe. De plus, il dispose d'un grand nombre de bibliothèques incluses ou à télécharger simplement permettent d'arriver très rapidement à un résultat concret.

1.2.2. Java et IDE NetBeans⁷

NetBeans est une plate-forme de développement logiciel en Java. La plate-forme NetBeans permet aux applications d'être développées à partir d'un ensemble de composants logiciels modulaires appelés modules. Les applications basées sur la plate-forme NetBeans, y compris l'environnement de développement intégré (IDE) NetBeans, peuvent être étendues par des développeurs tiers.

NetBeans s'exécute sur Microsoft Windows, Mac OS X, Linux, Solaris et d'autres plates-formes prenant en charge une JVM compatible.

⁷ <https://en.wikipedia.org/wiki/NetBeans>

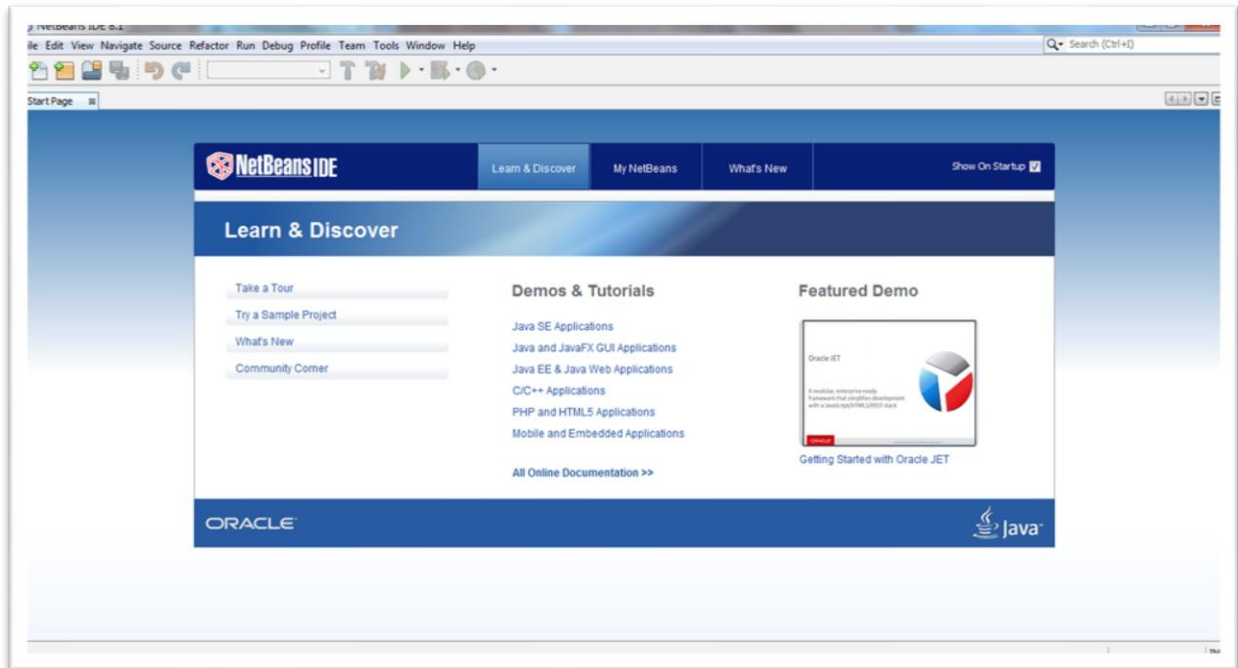


Figure 11. Interface de Netbeans IDE 8.2

1.2.3. Jython

Jython, anciennement nommé JPython, est un interprète Python écrit en Java, créé en 1997 par Jim Hugunin. Jython permet la compilation de code Python en bytecode Java, ainsi qu'un héritage de classes Java par des classes Python. Il permet aussi, l'exécution de code Python durant le fonctionnement d'un programme Java (scripting) et l'utilisation d'objets Java dans le code Python. [27]

2. Architecture générale du système

2.1. Le Crawling

C'est la première partie de système où l'utilisateur peut lancer le crawling au lancement du test de vulnérabilités.

Ainsi le crawler proposé spécialement pour détecter les failles de sécurité parcourt les différents liens de l'application afin de découvrir les chemins. Le crawler découvre les chemins possibles, les enregistre et les indexe afin de pouvoir les utiliser dans l'étape suivante.

2.2. Le scanner

Dans cette partie l'application recherche les vulnérabilités de l'application web à partir d'une liste des chemins et urls. Il scanne les pages Web déployées, recherche des scripts et de formulaires où il peut injecter des données. Une fois qu'il obtient cette liste, travaille injecte des charges utiles pour voir si un script est vulnérable et finalement, obtient ces attaques puis l'affiche.

2.3. Les résultats

C'est la dernière étape de l'application. Elle consiste à afficher les résultats correspondants à la requête d'utilisateur. Les résultats sont aussi interprétés afin d'aider à déterminer si la sécurité de l'application web est bien sécurisée ou non.

3. implémentation des composants du système

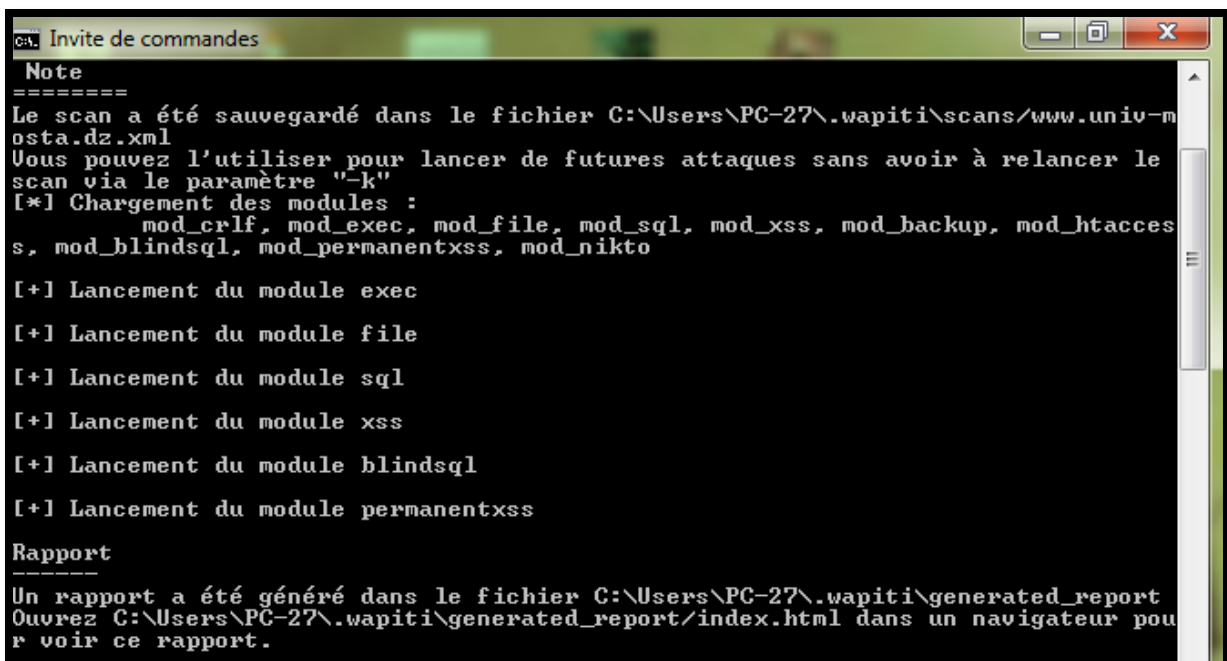
- ✓ lancer le test de vulnérabilité

L'utilisateur lance le programme avec l'URL de l'application web pour faire l'analyse et détecter les vulnérabilités.

```
C:\>cd scanner
C:\scanner>cd bin
C:\scanner\bin>python scanner www.univ-mosta.dz
Scanner
```

Figure 12. Lancement du Test de vulnérabilités

- ✓ Puis on obtient les résultats du test dans un dossier local comme suit :



```
CA Invite de commandes
Note
=====
Le scan a été sauvegardé dans le fichier C:\Users\PC-27\.wapiti\scans/www.univ-mosta.dz.xml
Vous pouvez l'utiliser pour lancer de futures attaques sans avoir à relancer le scan via le paramètre "-k"
[*] Chargement des modules :
    mod_crlf, mod_exec, mod_file, mod_sql, mod_xss, mod_backup, mod_htaccess, mod_blindsql, mod_permanentxss, mod_nikto
[+] Lancement du module exec
[+] Lancement du module file
[+] Lancement du module sql
[+] Lancement du module xss
[+] Lancement du module blindsql
[+] Lancement du module permanentxss
Rapport
-----
Un rapport a été généré dans le fichier C:\Users\PC-27\.wapiti\generated_report
Ouvrez C:\Users\PC-27\.wapiti\generated_report/index.html dans un navigateur pour voir ce rapport.
```

Figure 13. Fin du test de vulnérabilités

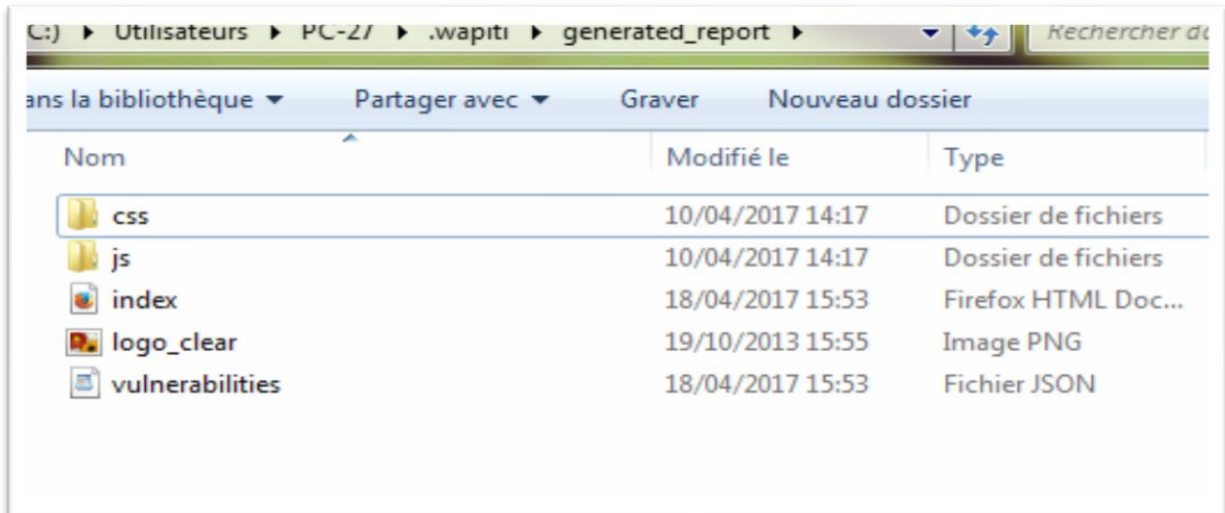


Figure 14. Les résultats du test

- ✓ Le résultat du crawling obtenu est sauvegardé dans un fichier XML. (voir figure suivante)

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <rootURL>http://www.univ-mosta.dz/</rootURL>
  <toBrowse/>
  <browsed>
    <resource encoding="UTF-8" method="GET" path="http://www.univ-mosta.dz/">
      <referer></referer>
      <get_params/>
      <post_params/>
      <file_params/>
      <headers>
        <header name="Content-Type" value="text/html"/>
        <header name="Transfer-Encoding" value="chunked"/>
        <header name="Connection" value="keep-alive"/>
        <header name="Keep-Alive" value="timeout=15"/>
        <header name="Date" value="Tue, 09 Apr 2017 23:18:54 GMT"/>
        <header name="Server" value="Apache"/>
        <header name="X-Powered-By" value="PHP/5.5.38"/>
        <header name="status_code" value="200"/>
      </headers>
    </resource>
    <resource encoding="windows-1252" method="GET" path="http://www.univ-mosta.dz/styles.css">
      <referer>http://www.univ-mosta.dz/</referer>
      <get_params/>
      <post_params/>
      <file_params/>
      <headers>
        <header name="Content-Type" value="text/css"/>
        <header name="Content-Length" value="14610"/>
        <header name="Connection" value="keep-alive"/>
        <header name="Keep-Alive" value="timeout=15"/>
        <header name="Date" value="Tue, 09 Apr 2017 23:18:54 GMT"/>
        <header name="Server" value="Apache"/>
        <header name="Last-Modified" value="Mon, 22 Jun 2015 20:42:10 GMT"/>
        <header name="ETag" value="&quot;3912-519214ec98c80&quot;"/>
        <header name="Accept-Ranges" value="bytes"/>
        <header name="status_code" value="200"/>
      </headers>
    </resource>
  </browsed>
</root>
```

Figure 15. Résultats du Crawling

- ✓ Pour l'affichage du résultat final, nous avons intégré une interface graphique qui permet de visualiser les résultats finaux et leurs interprétations.
- ✓ L'interface principale de notre application est illustrée sur l'image suivante :

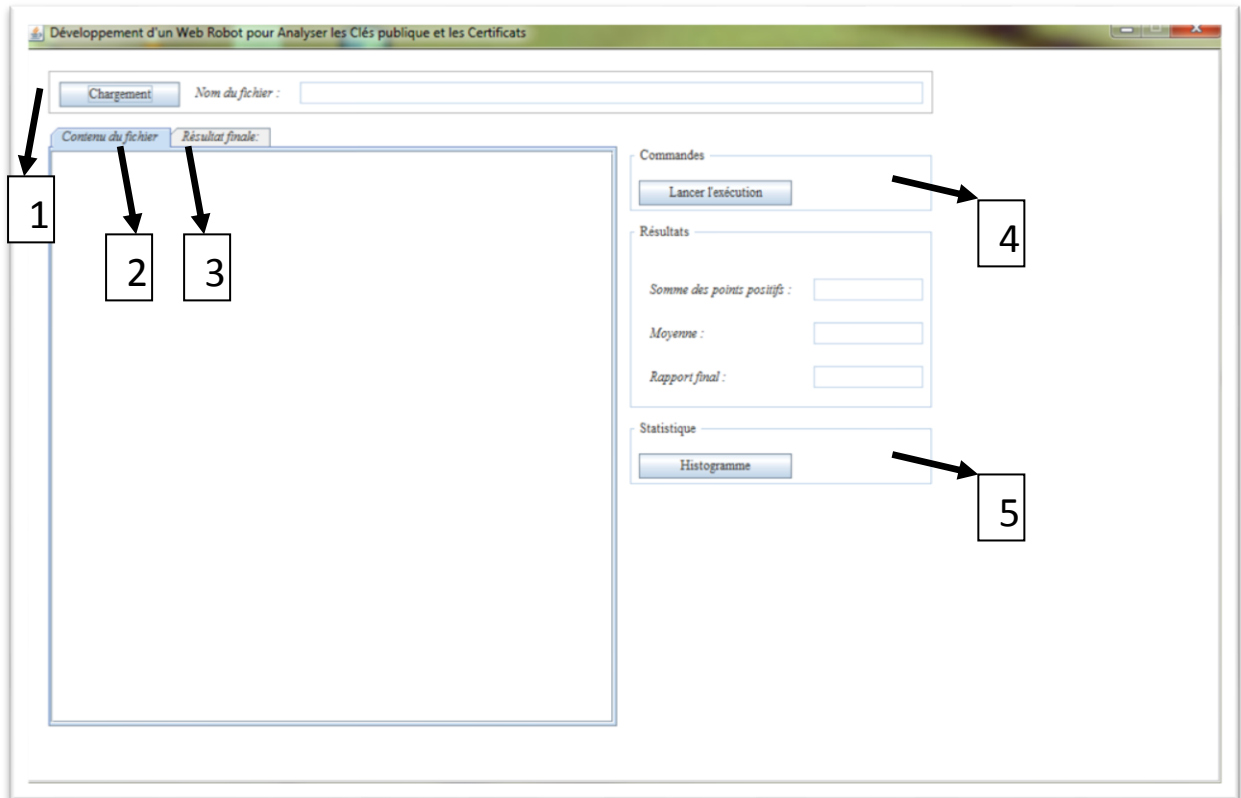


Figure 16. Interface Graphique de l'application

- 1-Chargement :** c'est pour charger le fichier qui contient la résultat final de l'exécution.
- 2-Contenu du fichier :** affiche le contenu du résultat de l'exécution.

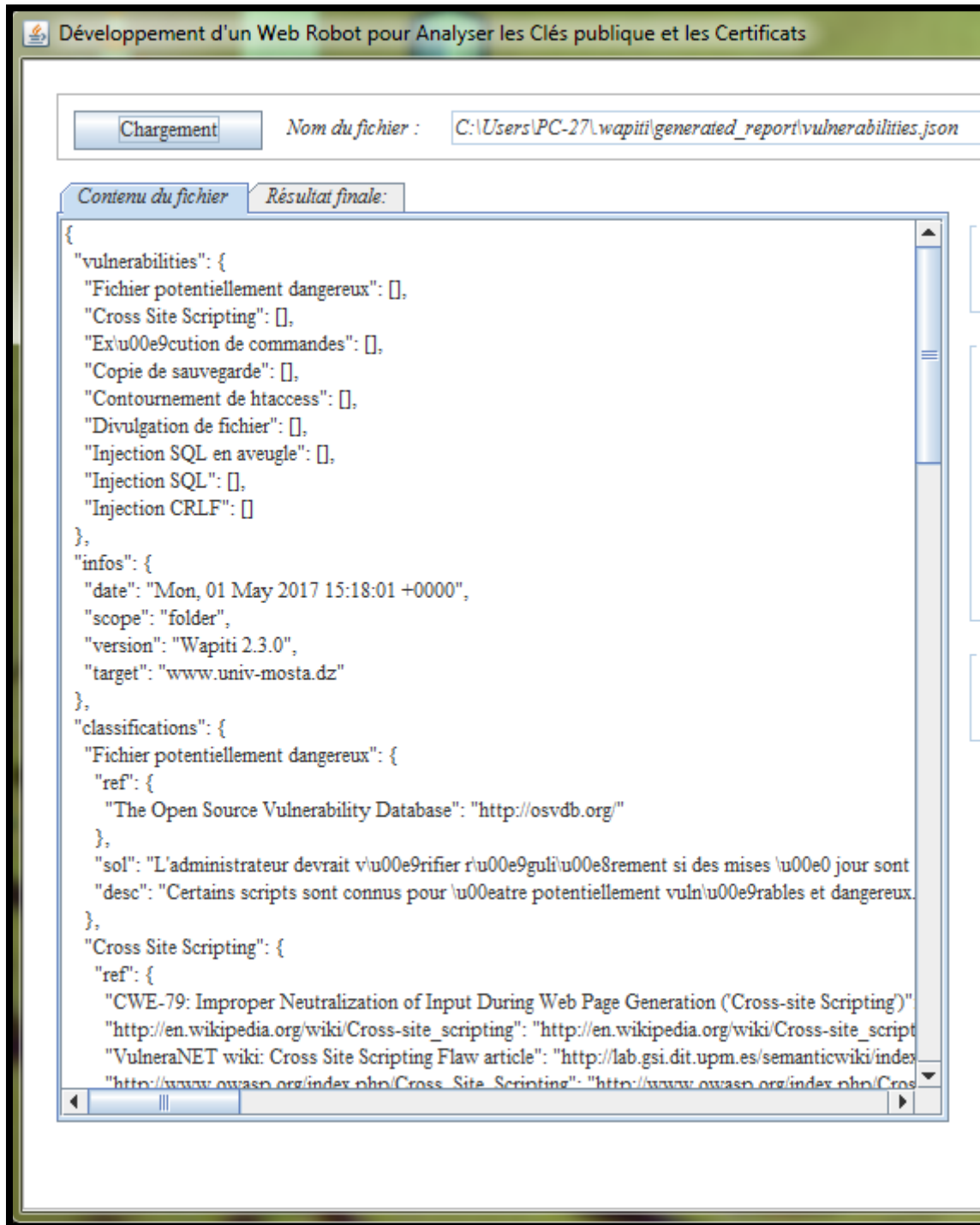


Figure 17. Contenu du fichier

3-Résultat final : c'est l'affichage de la liste qui indexe les points de vulnérabilité pour voir quels sont les points forts et faibles pour l'application web.

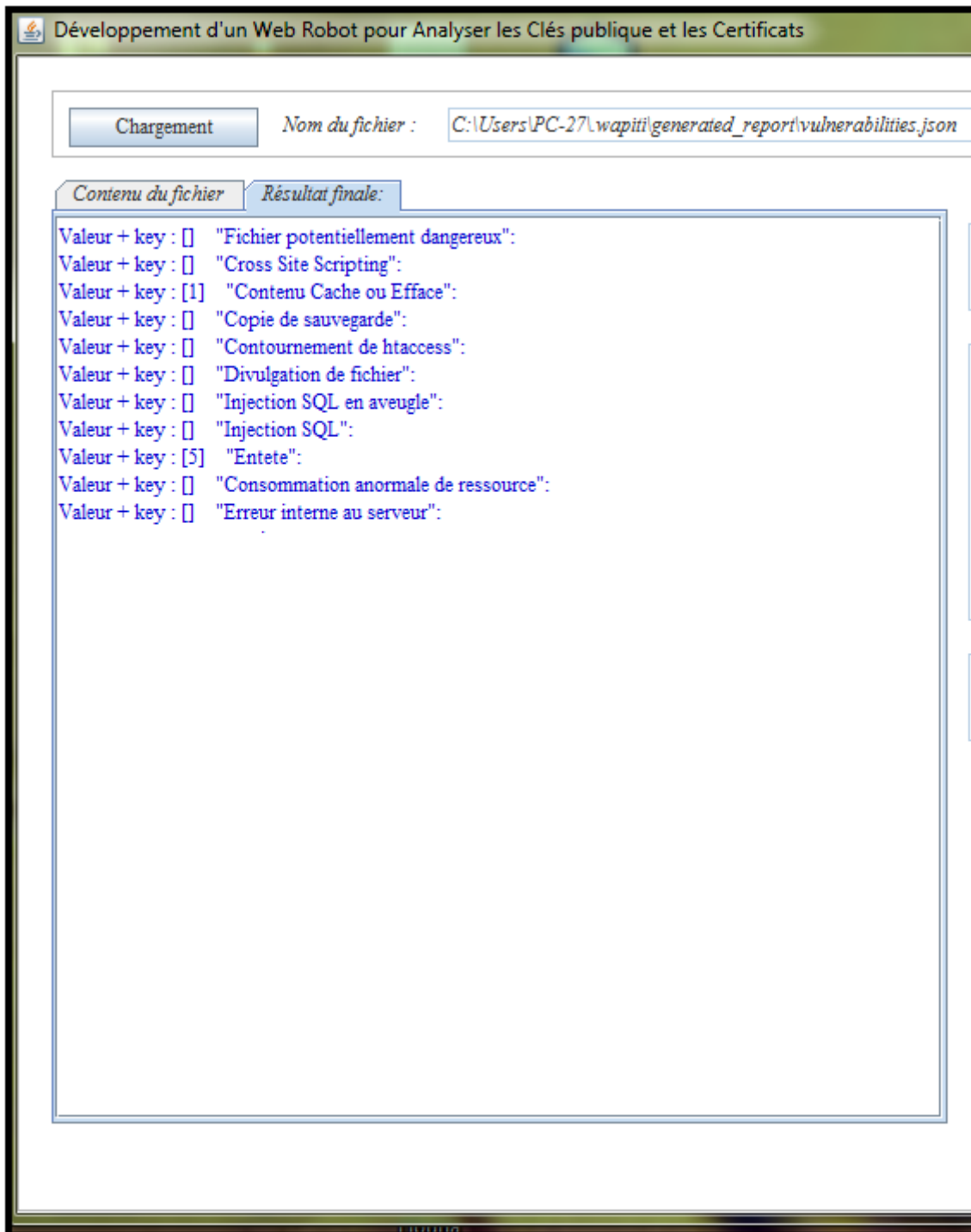


Figure 18. Résultat Final

4- Lancer l'exécution : Cette affichage récapitule les points de sécurité proposé pour notre travail après l'analyse, puis la somme des points de sécurité et la moyenne pour ces dernier points. D'après les résultats obtenus, le résultat pour l'application web peut être mal, bien, très bien sécurisée.

Commandes

Lancer l'exécution

Résultats

Somme des points positifs : 9

Moyenne : 8.181818181818182

Rapport final : Page web très bon securise

Statistique

Histogramme

Figure 19. Affichage de l'exécution

5-Histogramme : c'est l'historgramme des points des sécurités.

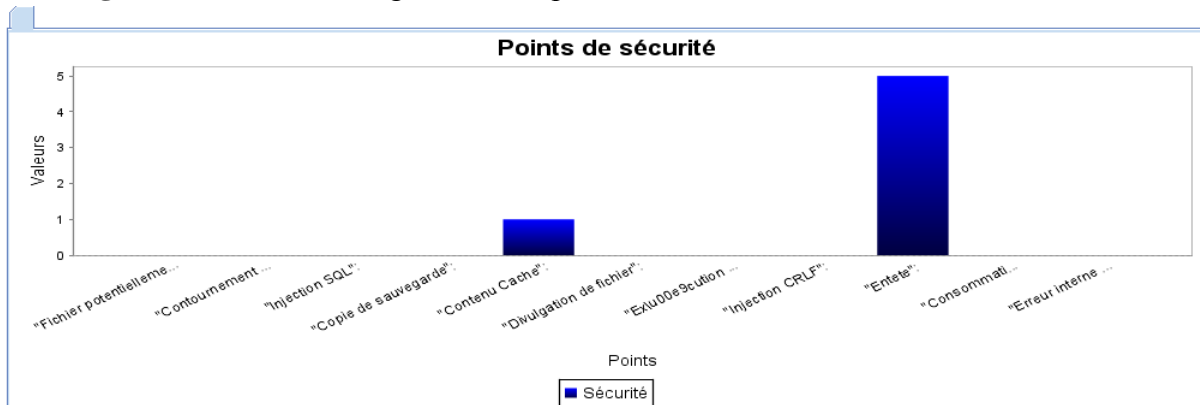


Figure 20. Histogramme des points des sécurités

Conclusion

Nous avons vu - tout au long de ce chapitre – les différents outils nécessaires pour la réalisation de notre système tels que l'environnement de développement, les langages de programmations.

Nous avons montré l'implémentation de notre système proposé, afin d'assurer une présentation claire et détaillée de notre outil. De plus, on a cité et expliqué les caractéristiques de notre application.

Finalement, nous avons donné quelques résultats obtenus on utilisant des captures d'écran.

Conclusion générale

De nos jours, les applications Web sont à la fois beaucoup plus répandues et beaucoup plus complexes. Le développement du Web dynamique et la richesse fonctionnelle qu'offrent les nouvelles technologies du Web permettent de répondre à un grand nombre de besoins.

L'objectif de ce travail était d'apporter des contributions sur le développement d'un Web-Robot qui va se charger d'auditer la sécurité d'une application web en testant différentes attaques comme l'aurait fait un pirate. Le travail présenté dans ce mémoire essaye de bénéficier au maximum des notions de robot web, et plus précisément côté sécurité.

La réalisation de travail a été divisée en trois parties :

- partie 1 : nous avons abordé les concepts et notions sur les robots d'indexation et le processus de crawling. Nous avons aussi présenté les différents problèmes à considérer pour développer un web crawler et nous avons abordé les applications web en nous focalisant sur l'aspect de la sécurité et vulnérabilité des applications web. Plus précisément, nous avons défini les différents concepts de la sécurité, Comme il est indiqué dans le chapitre 1.
- partie 2 : nous avons discuté de l'ensemble des techniques de protection pour les applications Web et nous nous focalisons principalement sur les scanners de vulnérabilités Web et l'utilisation des crawlers pour la détection. Ensuite, nous avons proposé une solution au problème posé, nous avons aussi détaillé les divers besoins de sécurité nécessaire au développement et conception de notre application. En dernier, nous avons abordé la modélisation statique et dynamique du système proposée. Comme il est indiqué dans le chapitre 2.
- partie 3: Nous avons exposé l'implémentation de l'application proposée en comme il est indiqué dans le chapitre 3.

Bien que notre évaluation initiale démontre la promesse d'une approche d'analyse de vulnérabilité, elle met en évidence plusieurs limitations qui constituent le fondement des travaux futurs dans ce domaine. Tout d'abord, nous ne fournissons pas le correctif pour l'URL vulnérable détectée. Une perspective future serait de fournir des solutions aux vulnérabilités détectées. Ensuite, un large domaine pour les travaux futurs serait d'explorer d'autres vulnérabilités, les détecter et fournir des solutions.

Références Bibliographiques

- [1] **Megague Khadidja**. Réalisation et implémentation d'un crawler performant dans le web. Informatique. 15 décembre 2014. 118 pages.
- [2] **Antoine Billard, et al.** Scraping & Crawling (Collecter et exploiter les données du Web). INSA Lyon. 22 Pages.
- [3] **Nguyen Hong San**. Module d'extraction focalise et analyse automatique linguistique du web. Informatique. Janvier 2007. 69 pages.
- [4] **Marios D.Dikaiakos, et al** . An investigation of web crawler behavior: characterization and metrics . Computer Communications 28, 2005, 880–897. 18 Pages.
- [5] **Trupti V. Udapure**. Study of Web Crawler and its Different Types. IOSR Journal of Computer Engineering (IOSR-JCE) 1, 16. 2014. Pages 1-5.
- [6] **Sotiris Batsakis, et al.** Improving the performance of focused web crawlers. Data & Knowledge Engineering. Data & Knowledge Engineering 68, 10. 2009. Pages 1001-1013.
- [7] **Filippo Menczer, et al.** Topical Web Crawlers: Evaluating Adaptive Algorithms. ACM Transactions on Internet Technology, Vol. 4, No. 4, 2004. Pages 378-419.
- [8] **Rahul Kumar et al.** Survey of web crawling algorithms. Advances in Vision Computing: An International Journal (AVC) Vol.3, No.3, Sep 2016. Pages 1-7.
- [9] **Pavalam Sm et al.** A Survey of Web Crawler Algorithms. IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 6, No 1, November 2011. Pages 5.
- [10] **Web Application Security Consortium. Web Application Security Scanner Evaluation Criteria.** Version 1. 2009. Pages 1-26.
- [11] **Oracle.** Understanding Web Service Security Concepts. [En ligne]. Disponible à l'adresse : <https://docs.oracle.com/middleware/1212/owsm/OWSMC/owsm-security-concepts.htm>. Consultée le : 10/12/2016.
- [12] **Amyr Bennamane.** Cryptographie Le protocole SSL. Informatique –Cryptographie. 16 mai 2009. Pages 14.
- [13] **Chabane Refes.** Les Services Web. [En ligne]. Disponible à l'adresse : <https://openclassrooms.com/courses/les-services-web>. Consultée le : 5/12/2016.
- [14] https://www.owasp.org/index.php/Main_Page . Consultée le : 20 / 03/2017.
- [15] https://www.owasp.org/index.php/Category:OWASP_Testing_Project. Consultée le : 20 /03 /2017.
- [16] <https://openclassrooms.com/courses/debutez-l-analyse-logicielle-avec-uml/uml-c-est-quoi>. Consultée le : 30 /03 /2017.
- [17] <http://www.uml-sysml.org/diagrammes-uml-et-sysml/diagramme-uml/structurel-ou-statique>. Consultée le : 30 /03 /2017.

- [18] <http://www.uml-sysml.org/diagrammes-uml-et-sysml/diagramme-uml/diagrammes-comportementaux> Consultée le : 30 /03 /2017.
- [19] <https://larlet.fr/david/biologeek/archives/20060505-pourquoi-programmer-en-python/> Consultée le : 15/04 /2017.
- [20] <https://en.wikipedia.org/wiki/NetBeans> Consultée le : 15/04 /2017.
- [21] https://fr.wikipedia.org/wiki/Java_Development_Kit Consultée le : 15/04 /2017.
- [22] <https://en.wikipedia.org/wiki/Jython> Consultée le : 15/04 /2017.
- [23] <https://openclassrooms.com/courses/debutez-l-analyse-logicielle-avec-uml/uml-c-est-quoi> Consultée le : 15/04 /2017.
- [24] <http://www.uml-sysml.org/diagrammes-uml-et-sysml/diagramme-uml/structurel-ou-statique> Consultée le : 30 /03 /2017.
- [25] <http://www.uml-sysml.org/diagrammes-uml-et-sysml/diagramme-uml/diagrammes-comportementaux> Consultée le : 30 /03 /2017.
- [26] <https://larlet.fr/david/biologeek/archives/20060505-pourquoi-programmer-en-python/> Consultée le : 16/04 /2017.
- [27] <https://en.wikipedia.org/wiki/Jython> Consultée le : 15/04 /2017.