الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université de 8 Mai 1945 – Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière
Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Système Informatique

Thème :

La détection des webshells pour assurer la sécurité des serveurs web

Encadré par :

Dr. HANNOUSSE ABDELHAKIM

Présenté par :

AID MOHAMED AMINE

Septembre 2021

Remerciement

Nous tenons à la fin de ce mémoire de remercier avant tout ‘’ALLAH’’ le tout puissant, de m’avoir donné le courage, la grande volonté et la patience pour l’accomplissement à temps de ce modeste travail.

Je remercie aussi tous ceux qui m’ont aidé de près ou de loin chacun par son nom, à la confection de ce modeste mémoire.

D’abord je remercie chaleureusement mon directeur de recherche
Monsieur **HANNOUSSE ABDELHAKIM** pour ses conseils judicieux, son encouragement et sa persévérance.

Je remercie aussi ma tante **BESNACI LEILA** qui m’a montré le chemin d’un travail de qualité.

Mes remerciements les plus vifs s’adressent également aux membres de jury pour l’intérêt qu’ils ont porté à ma recherche tout en acceptant d’évaluer mon travail.

Enfin, je tiens à remercier infiniment tous les enseignants et les étudiants du département de l’informatique.

Dédicace

J’ai le plaisir de dédier ce modeste travail :

À mes chers parents « **LAHCEN et CHERICHERI HADDA** », qui ont veillé sur moi et qui m’ont profondément aidé durant mes études par leurs sacrifices et leurs prières. Qu’ALLAH le tout puissant leur procure une longue vie.

À mon frère « **CHAMSSOU** » et à ma sœur « **MALIKA** » qui ont partagé avec moi tous les moments d’émotion lors de la réalisation de ce travail. Merci beaucoup pour votre soutien.

À vous aussi mes chers amis « **RAOUF**, **ABDELMOUMEN** » et d’autres, chacun par son nom avec qui j’ai partagé les meilleurs et les agréables moments de mon parcours universitaire.

À tous ceux qui m’ont aidé matériellement et moralement et à ceux qui m’ont donné de la force pour réaliser mon projet dans de bonnes conditions.

RESUME

Ces dernières années, il y a eu une augmentation des attaques webshells. Les webshells sont des scripts qui peuvent être écrits dans différents langages, notamment PHP. Ils sont téléchargés sur des serveurs Web après avoir créé une brèche en utilisant des vulnérabilités d'injection. Les webshells fournissent aux attaquants une interface Web pour exécuter des commandes à distance, manipuler des données confidentielles et envahir les serveurs Web. Les systèmes classiques de détections des fichiers contenant des scripts malveillants comme les antivirus et les firewalls sont devenus obsolètes. Ces logiciels sont dans la plupart du temps inefficaces face à l'évolution de nouvelles attaques plus sophistiquées. Dans cette étude, nous examinons l'efficacité de l'apprentissage automatique dans la détection des webshells. Notre objectif est d'implémenter un modèle d'apprentissage profond efficace pour la détection des webshells écrits en PHP. Nous évaluons la performance du modèle proposé avec d'autre modèles d'apprentissage traditionnels. Nous développons une application permettant d'exploiter notre modèle en pratique. L'application permet d'analyser et d'identifier automatiquement les fichiers suspects situés dans un emplacement spécifié. Elle permet aussi aux utilisateurs de sauvegarder la prédiction de chaque modèle afin de comparer leurs performances.

Mots-clés: détection des webshells, sécurité sur Internet, apprentissage automatique.

ABSTRACT

In recent years, there has been an increase in webshell attacks. Webshells are scripts that can be written in different languages, including PHP. They are uploaded to web servers after creating a breach using injection vulnerabilities. Webshells provide attackers with a web interface to execute remote commands, manipulate confidential data, and invade web servers. Traditional systems for detecting files containing malicious scripts, such as antivirus programs and firewalls, have become obsolete. These software are mostly ineffective in the face of the development of new, more sophisticated attacks. In this study, we examine the effectiveness of machine learning in detecting webshells. Our goal is to implement an efficient deep learning model for the detection of webshells written in PHP. We evaluate the performance of the proposed model with other traditional learning models. We are developing an application to use our model in practice. The application can automatically scan and identify suspicious files located in a specified location. It also allows users to save the prediction of each model in order to compare their performances.

Keywords: webshell detection, Internet security, machine learning.

ملخص

في السنوات الأخيرة، كانت هناك زيادة كبيرة في هجمات الويب شال. حيث الويب شال هي نصوص يمكن كتابتها بلغات برمجة مختلفة بما ذلك PHP.
حيث يتم تحميلها على خوادم الويب بعد انشاء خرق باستخدام نقاط ضعف الحقن الموجودة في الخادم. توفر الويب شال للمهاجمين واجهة ويب لتنفيذ الأوامر عن بعد، والتلاعب بالبيانات السرية وغزو خوادم الويب.
الانظمة الكلاسيكية لاكتشاف الملفات التي تحتوي على نصوص خبيثة مثل مضادات الفيروسات وجدران الحماية قد أصبحت قديمة.
هذه البرامج غير فعالة في معظم الأحيان في مواجهة تطور الهجمات الأكثر تعقيدا.
في هذه الدراسة، ندرس فعالية التعلم الآلي في اكتشاف الويب شال. هدفنا هو تنفيذ نموذج للتعلم العميق الفعال للكشف عن الويب شال المكتوبة بلغة PHP.
نقوم بتقييم أداء النموذج المقترح باستخدام نماذج التعلم التقليدية الأخرى. نحن نعمل على تطوير تطبيق يسمح لنا باستخدام نماذجنا في الممارسة.
هذا التطبيق يسمح بتحليل وتحديد تلقائياً الملفات المشبوهة التي تقع في مكان محدد في الخادم. كما يسمح للمستخدمين بحفظ تنبؤات كل نموذج لمقارنة أدائهم.

الكلمات الرئيسية: الكشف عن الويب شال، أمن الانترنت، التعلم الآلي.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

INTRODUCTION GENERALE

Depuis l'apparition d'Internet, le nombre d'utilisateurs des services web a connu une augmentation phénoménale. Les serveurs web peuvent traiter un nombre énorme de requêtes envoyées depuis des machines à distance en utilisant le protocole HTTP. Ce dernier est considéré comme le protocole le plus utilisé dans le Web afin d'échanger des données avec des serveurs distribués. L'augmentation de l'utilisation des services web a entrainé l'augmentation de l'utilisation malveillante de ses services ; à travers l'utilisation de plusieurs méthodes et techniques, les hackers peuvent enchainer des attaques sur les serveurs web. L'une des attaques les plus dangereuses qui cible les serveurs web est l'attaque webshell.

Les webshells sont des scripts qui permettent aux attaquants d'accéder et de manipuler à distance des serveurs web depuis des navigateurs web réguliers. En assurant l'accès à ces scripts malveillants à distance, les attaquants peuvent réaliser un ensemble d'opérations malveillantes telle que l'exploitation des informations sensibles et le lancement d'autres attaques plus dangereuses. Pendant la création des webshells, les attaquants utilisent plusieurs langages de scripts ; les études récentes ont montré que PHP est le langage le plus utilisé pour la création de ces scripts malveillants.

Pour lutter contre ce type d'attaques, plusieurs efforts ont été réalisés pour le développement des outils qui permettront de détecter les webshells malveillants écrits en langage PHP. Dans ce cadre s'inscrit notre projet de fin d'étude, qui s'intéresse à examiner un ensemble de modèles se basant sur les technologies d'apprentissage automatique. Spécifiquement, nous proposons un modèle d'apprentissage profond et comparons l'efficacité du modèle proposé avec des modèles d'apprentissage traditionnels. Nous examinons ainsi l'impact des combinaisons des modèles d'apprentissage profonds avec des modèles d'apprentissage traditionnels. A la fin, nous développerons une application qui mettra en pratique notre modèle et permettra l'analyse des fichiers situés dans un chemin spécifié par l'utilisateur et qui pourra

détecter la présence ou pas des webshells de type PHP. Pour réaliser cet objectif, le présent mémoire est structuré en cinq chapitres :

Chapitre 1 : dans ce premier chapitre, nous définissons le concept des webshells et nous expliquons ces différents types et le principe de base de ces attaques.

Chapitre 2 : dans ce chapitre, nous montrons les principes d'apprentissage automatique en se concentrant sur l'apprentissage profond, qui est la technologie de base utilisé par notre solution proposée.

Chapitre 3 : on s'intéresse, dans ce chapitre, à présenter les différentes solutions récentes proposées pour la détection des webshells qui se basent sur l'apprentissage automatique. Nous détaillons quelques solutions proposées dans ce contexte. Cette présentation nous permettra d'avoir un aperçu sur les derniers développements réalisés pour la détection des webshells par l'utilisation de l'apprentissage automatique.

Chapitre 4 : dans ce chapitre, nous décrivons l'étude conceptuelle de notre solution proposée pour la détection des webshells de type PHP. Cette proposition se base sur l'utilisation d'un ensemble de modèles d'apprentissage automatique, qui se subdivisent en modèles de base et en modèles comparatifs dans le but d'obtenir le modèle le plus performant.

Chapitre 5 : dans ce dernier chapitre, nous présentons la phase de conception et de développement d'un outil nommé « *PHP Webshell Scanner* ». Ce dernier permet aux utilisateurs de lancer plusieurs modèles d'apprentissage automatique, inclus dans notre modèle proposé, pour détecter les webshells qui résident sur un chemin préalablement spécifié. L'outil permet aussi de sauvegarder les prédictions de chaque modèle choisi afin de faire des comparaisons.

CHAPITRE I.

INTRODUCTION AUX WEBSHELLS

CHAPITRE I.

INTRODUCTION AUX WEBSHELLS*

Les serveurs Web sont indispensables pour diffuser des informations et des services sur Internet ou Intranet. Grâce aux serveurs Web, il est possible d'enregistrer des contenus et des services Web et d'assurer leurs accessibilité permanente aux utilisateurs à distance. Cependant, l'expérience a prouvé que la sécurité des serveurs Web reste un problème critique pour toute organisation, publique ou privée, qui veut mener ses services en ligne. Les serveurs Web d'aujourd'hui hébergent un grand nombre de données et services dont certains sont très sensibles ce qui les rend plus ciblés par les cyberattaques. La sécurisation d'un serveur Web est essentielle pour assurer la confidentialité des données hébergées d'une part et le bon fonctionnement et l'accessibilité des services offerts aux utilisateurs d'une autre part.

Dans le présent chapitre, nous nous intéressons à l'une des plus dangereuses et plus fréquentes attaques observées ces dernières années, qui cible les serveurs Web : *l'attaque Webshell*. Dans ce chapitre, nous présenterons en détails le principe de cette attaque, ses différents types et les difficultés empêchant leur détection.

1.1. Définition et popularité de l'attaque Webshell

Avant d'explorer les détails de l'attaque Webshell, il faut d'abord donner un aperçu sur la définition du Webshell. En réalité, le terme Webshell est composé de deux mots anglais : *Web* qui signifie la toile d'araignée mondiale, et *Shell* qui se traduit par coquille en français. Pour être plus précis, Webshell est un terme spécifique indiquant un script malveillant permettant l'accès et le contrôle à distance des serveurs à travers un navigateur Web. Ce terme peut se traduire en français par *code encoquillé*. Dans ce mémoire nous adoptons la nomenclature anglaise de ce terme pour sa simplicité.

* Le présent chapitre est en commun avec trois étudiants en Master de l'année 2021, travaillant sur le Webshell, un étudiant de l'université 8 Mai 1945 Guelma, dirigé par Dr Abdelhakim Hannousse, et deux étudiants de l'université Badji Mokhtar Annaba, dirigés par Mme Salima Yahiouche.

Un Webshell assure une porte dérobée (*backdoor* en anglais) dans une page web, qui permet aux attaquants de contrôler le serveur Web à distance en perpétrant des opérations non autorisées telles que : la collecte d'informations, la modification des paramètres d'authentification et des privilèges d'accès aux données, l'exécution des commandes et la manipulation des fichiers. En outre, une attaque Webshell peut se reproduire en infectant tous les autres serveurs connectés au serveur ciblé, ce qui la rend encore plus redoutable. Malgré la disponibilité d'une gamme importante des logiciels et systèmes de protection et de détection des malwares comme les antivirus, les attaques Webshell restent redoutables dans le monde du Web. La figure 1.1 montre le nombre des attaques Webshells enregistrées par l'équipe de recherche Microsoft 365 Defender dans la période de Août 2019 à Janvier 2021. La figure indique une croissance remarquable pour l'année 2020 et le début de l'année 2021. Contrairement aux autres types des malwares tels les virus, le Webshell se caractérise par :

1. Un script passif qui ne fonctionne qu'à l'intervention et au moment de l'accès de l'attaquant.
2. Il s'exécute exclusivement sur un navigateur Web.
3. Une attaque persistante qui peut durer longtemps et passer inaperçue devant la vigilance d'un administrateur.

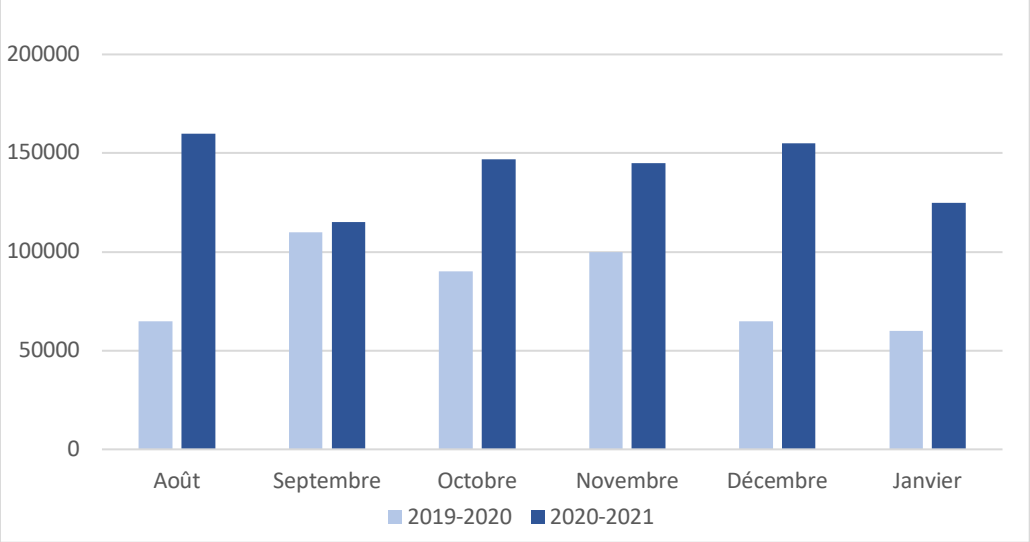

Figure 1.1. Le nombre d'attaques Webshell enregistrées de 2019 à 2021 [W1]

1.2. Langages utilisés pour le codage des Webshells

Le code des Webshells est généralement écrit dans un des langages de script telles PHP, ASP, ASPX, JSP et PERL-CGI. Pour éviter que les Webshells ne soient facilement détectés, ils sont généralement programmés par les mêmes langages utilisés pour les applications hébergées dans le serveur Web ciblé. Les dernières études sur l'utilisation des langages de script pour la programmation des applications coté serveurs ont montré que 21 langages de script sont utilisés. Les langages couramment utilisés sont PHP (79.2%) notamment ses deux versions ; version 7 (65.3%) et version 5 (34.1%). La figure 1.2 indique l'évolution de l'utilisation des différents langages scripts pour la programmation des applications coté serveur enregistrée jusqu'au 3 Juin 2021 par Web Technology Surveys (W3techs) [W2]. Cela donne un aperçu sur les langages des différentes attaques Webshell enregistrées durant cette période.

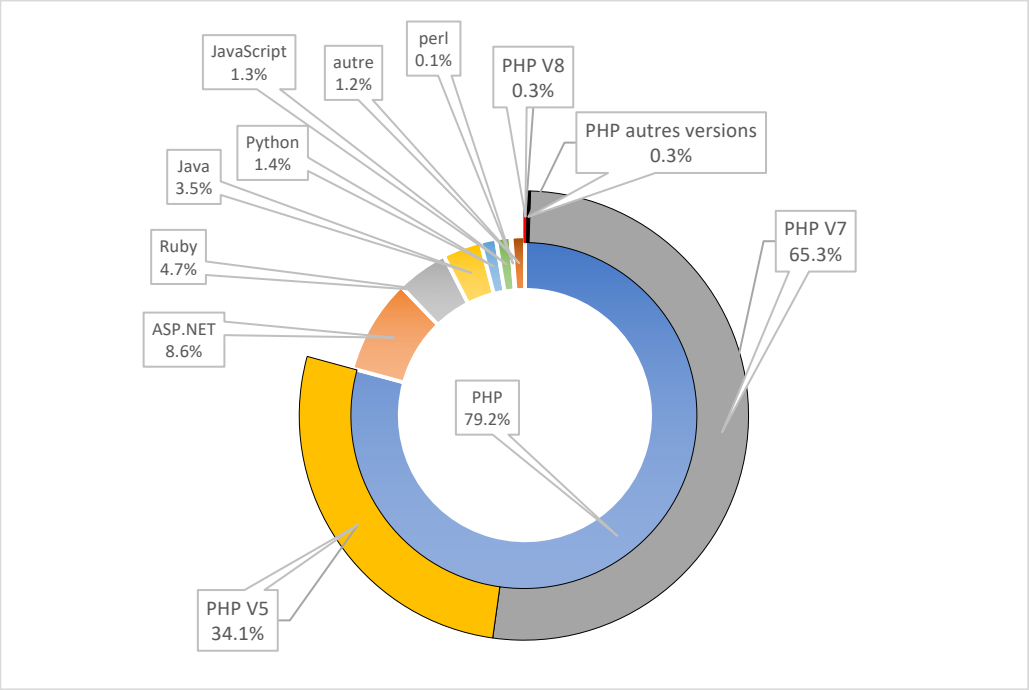

Figure 1.2. Les langages scripts utilisés pour la programmation des applications coté serveur jusqu'au 3 Juin 2021 [W2]

1.3. Classification des Webshells

Il existe des millions de Webshells différents et différentes classifications sont disponibles : classification par langage (e.g. ; Webshell PHP, Webshell ASP, etc.) [1], classification par taille (e.g.; petit, grand et à un mot) [2] et classification par familles (e.g. ; c99, r57, B374k, etc.) [3]. Une des classifications les plus adoptée est constituée de trois catégories selon le rôle et les fonctionnalités offertes par un Webshell : *Lanceur de commandes*, *Chargeur de fichiers* et *Polyvalent* [4].

1.3.1. Lanceur de commandes

Ce type des Webshells est très simple et court, il ne possède qu'une seule ligne. Ce dernier est utilisé pour recevoir des commandes soumises par l'attaquant dans la requête HTTP accédant à la page du Webshell. En conséquence, le Webshell exécute les commandes reçues en paramètre et affiche le résultat de l'exécution sur la page Web. Le script suivant montre un exemple de ce type de Webshells écrit en PHP [4] :

```
<?PHP eval($_POST ["cmd"]);?>
```

Par ce script, quand l'attaquant envoie la requête ***http://url_page_webshell?ls***, le paramètre *cmd* du script va capturer la commande ***ls*** qui sera exécutée par la fonction PHP *eval() ;* le résultat va être affiché sur la page Web. Ici, la commande Unix ***ls*** a pour rôle d'afficher le contenu du répertoire où se trouve le fichier du Webshell.

1.3.2. Chargeur de fichiers

Ce type de Webshells fournit à l'attaquant la possibilité de charger des fichiers aux serveurs. Il est généralement utilisé comme tremplin pour les Webshells polyvalents. Habituellement, les serveurs Web imposeront certaines restrictions sur la taille ou le type de fichiers à charger. L'attaquant utilise ce Webshell, pour transférer vers le serveur des fichiers avec une taille qui dépasse celle autorisée par ce dernier ; généralement, ce genre de fichiers peut être un Webshell polyvalent ou un autre type plus dangereux de malwares. Un exemple de code PHP pour ce type de Webshells est le suivant [4] :

```php
<?php
    if (isset ($_POST["upload"])) {
        @file_put_contents($_POST["path"], $_POST["content"]);
    }
?>
```

Par ce script, l'attaquant peut transférer un Webshell polyvalent et peut contourner les restrictions de taille imposées par le serveur. Le contenu et le chemin où le nouveau Webshell douvent être sauvegardé dans le serveur sont envoyés dans une requête HTTP via les deux paramètres *path* et *content* respectivement, suivant la même technique montrée pour l'exemple de de la section 1.3.1.

1.3.3. Webshell polyvalents

Les Webshells polyvalents sont de grande taille en raison de la richesse des fonctionnalités offertes aux attaquants. Ces fonctionnalités sont accessibles à travers des interfaces graphiques sophistiquées. La figure 1.3 montre un exemple de Webshell polyvalent appelé *CMSmap* – WordPress qui a été découverte par Malware Research Team en Janvier 2020.

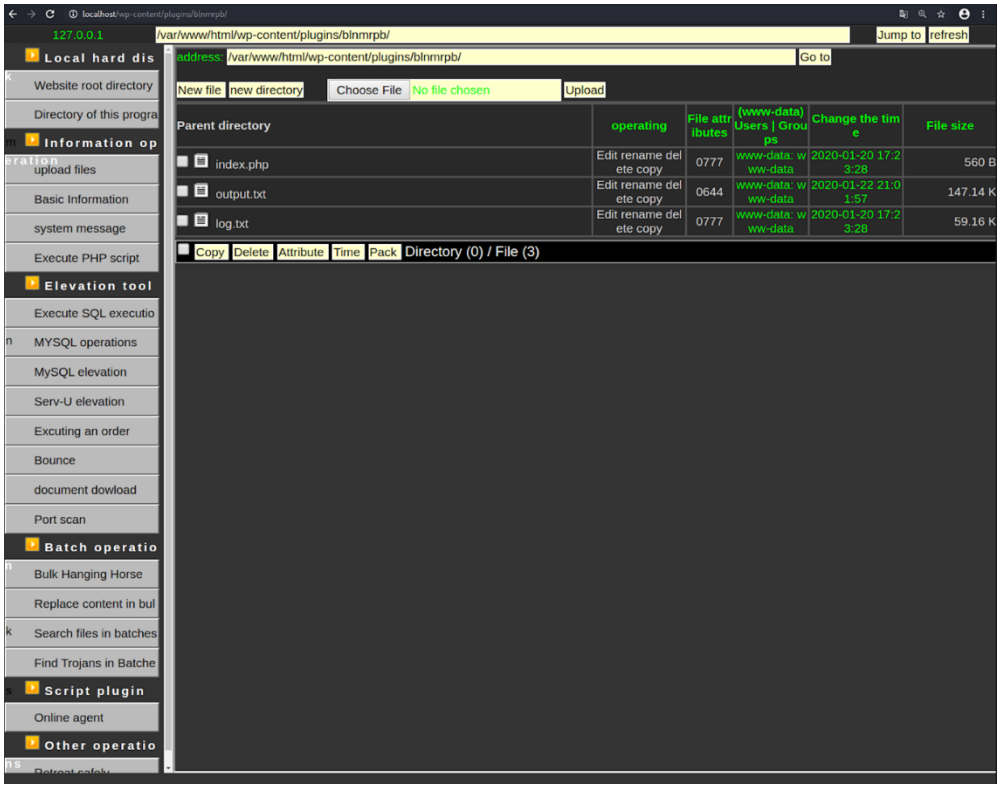

Figure 1.3. Interface du Webshell CMSmap - WordPress [W3]

1.4. Le Modèle d'attaque Webshell

Une attaque Webshell se réalise en trois étapes successives : *préparation*, *installation* et *exécution* comme la montre la figure 1.4. Dans ce qui suit nous détaillons chacune de ces étapes.

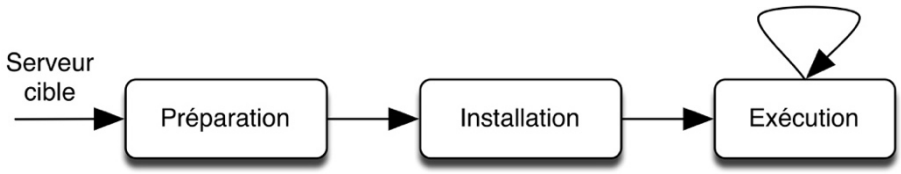

Figure 1.4. Modèle de l'attaque Webshell

1.4.1. Préparation de l'attaque

Après la détermination du serveur cible, les attaquants commencent par le codage du Webshell ; ils doivent connaitre le langage le plus utilisé dans ce serveur. En profitant de l'expressivité des langages scripts, les attaquants appliquent plusieurs techniques de codage qui permettent de dissimuler leurs activités malveillantes et contourner les administrateurs et les logiciels de protection installés sur le serveur. Dans cette section nous présentons quatre de ces techniques les plus utilisées : *découpage en lettres*, *cryptage, fraction de code en plusieurs pages* et *confusion de code*. À titre démonstratif, les quatre techniques présentées dans cette section sont appliquées au Webshell du lanceur de commandes présenté dans la section 1.3.1.

1.4.1.1. Découpage en lettres

Le découpage en lettres est une technique de dissimulation qui permet au code suspect d'un Webshell de passer inaperçu en profitant des fonctionnalités de codage offertes par le langage de script [1]. Le code d'origine du Webshell du type lanceur de commandes présenté dans la section 1.3.1 montre l'utilisation de la fonction PHP *eval()* pour exécuter ses opérations malveillantes. Une des techniques de dissimulation fréquemment utilisée est de subdiviser le nom de cette fonction en lettres et de sauvegarder chaque lettre dans une variable. Par la suite une nouvelle

variable est utilisée pour concaténer les valeurs des autres variables et former le nom de fonction désiré. Enfin, l'appel de la fonction se fait en appelant le nom de cette nouvelle variable comme le montre le script PHP suivant :

```php
<?php
        $v1 = "e";
        $v2 = "v";
        $v3 = "a";
        $v4 = "l";
        $v = $v1.$v2.$v3.$v4;
        $v($_POST["cmd"]);
?>
```

1.4.1.2. Cryptage de code

Le cryptage ou *chiffrement* est une technique très utilisée afin de rendre la compréhension du code des Webshells impossible sans avoir la clé de décryptage appropriée [1]. Différentes fonctions de cryptage sont disponibles en langages de scripts notamment le langage PHP. Le problème peut devenir plus complexe par l'utilisation de plusieurs niveaux de cryptage ou des fonctions de cryptages personnalisées. Le script suivant montre l'utilisation de la fonction de cryptage *base64_encode()* offerte par PHP pour crypter le code de notre exemple démonstratif. La fonction *base64_decode()* dans le script permet de décoder la chaine de caractères en paramètre avant que le script soit exécuté.

```php
<?php
    base64_decode( 'ZXZhbCAoX1BPU1QgWyJQIl0pOw== ');
?>
```

1.4.1.3. Fractionnent de code en plusieurs pages

Un Webshell peut ne pas contenir des signes d'activités malveillantes, mais il peut inclure des fichiers qui sont eux même nuisibles et téléchargeables à la demande. En PHP, les fonctions *include/require* peuvent être utilisées pour charger d'autres fichiers. Le script écrit ci-après, présente un exemple de Webshell incluant un fichier texte nommé *log.txt* ; ce script apparait légitime mais le fichier *log.txt* contient le code réel du Webshell (e.g.; le script de notre exemple démonstratif).

```php
<? php
    include('log.txt');
?>
```

En pratique, le fichier *log.txt* n'est pas analysé en tant que fichier PHP par le serveur, à moins que des modifications spécifiques ne soient apportées au fichier *.htaccess* ou à la configuration HTTP. Lorsque le fichier *log.txt* est chargé par la fonction *include()*, le contenu malveillant de *log.txt* peut être chargé, analysé et exécuté. Cette technique est utilisée par CMSmap – WordPress Webshell.

En outre, le nom du fichier intermédiaire spécifié dans la fonction *include()* peut aussi être placé à distance et récupéré comme paramètre dans la requête HTTP comme le montre le script suivant :

```
<? php
    $filename = $_POST['file'];
    include($filename);
?>
```

Cette technique est aussi utilisée pour découper le code d'un Webshell en plusieurs pages afin d'éviter trop d'appels de fonctions ou de variables suspectes dans un seul fichier. La page principale peut donc inclure les autres fichiers et appeler leurs fonctions ou variables. Cela réduit la proportion de fonctions ou de variables suspectes et peut ainsi réduire efficacement la probabilité d'être détecté [1].

1.4.1.4. Confusion de code

Cette technique consiste à insérer une grande quantité de commentaires, de variables ou de fonctions inutiles dans le code des Webshells pour cacher le code suspect et réduire sa proportion. De ce fait, les administrateurs vont mal comprendre le code et les logiciels de détection seront également contournés [1]. Le script suivant montre l'application de la technique de confusion sur notre exemple démonstratif.

```
<?php
    var/*-/*-*/v/*-/*-*/=/*-/*-*/"$_"+/*-/*-*/"P"+"O"+"S"+"T"+/*-/*-
      */"["+"'"+"cmd"/*-/*-*/+"'"+"]"; eval(/*-/*-*/v/*-/*-*/);
?>
```

1.4.2. Installation des Webshells

L'installation est l'étapes critique et la plus importante dans le modèle d'attaque des Webshells. Dans cette étape, l'attaquant se base sur l'exploration et l'exploitation des

failles de sécurité et des vulnérabilités du serveur cible pour intégrer et installer son Webshell. Parmi les grandes vulnérabilités connues, il y a le mauvais filtrage des entrées qui se fait au niveau des ports 80 et 443 [5, 6]. Ces ports sont responsables de recevoir des connexions via des requêtes HTTP ou HTTPS depuis les utilisateurs.

Une fois la pénétration réussite, les attaquants peuvent cacher leurs codes malveillants en les injectant dans des fichiers existant dans le serveur ou, charger directement le code dans un nouveau fichier avec un nom ordinaire (e.g.; *index.php*), habituellement placé dans un répertoire contenant de nombreux fichiers. L'attaquant doit se souvenir du chemin correct de l'emplacement pour lancer l'exécution du code malveillant.

1.4.3. Lancement des Webshells

Après installation, le Webshell installé devient prêt à être exécuté. Vu la nature du Webshell, son comportement ne s'exécute que via l'accès de l'attaquant. Les attaquants peuvent être connectés à leurs Webshells installés à travers un navigateur Web comme n'importe quel utilisateur qui veut se connecter à Internet, pour réaliser des accès légitimes aux données. La connexion à un Webhsell installé peut être réalisée de deux manières selon le type du Webshell :

1. Pour les lanceurs de commandes et les chargeurs de fichiers, la connexion et l'exécution se font en même temps. Il suffit que l'attaquant indique dans la barre d'adresse du navigateur le chemin correct où le Webshell est installé suivi par une séquence de paramètres requises, le Webshell reçoit alors ces données et exécute son comportement suivant les valeurs des paramètres reçus.

2. Pour les Webshells polyvalents, la connexion se fait par l'accès à la page du Webshell par le navigateur. L'interface graphique du Webshell sera affichée permettant à l'attaquant de choisir le type de l'opération malveillante à exécuter sur le serveur.

1.5. Conclusion

Dans ce chapitre, nous avons présenté quelques concepts de base concernant le Webshell : une attaque passive qui peut endommager un serveur web via des scripts. Nous avons commencé par donner quelques définitions, ensuite une classification selon la fonctionnalité du Webshell : lanceur, chargeur et polyvalent. Nous avons enfin présenté un modèle d'attaque basique pour le Webhsell incluant les phases de préparation, d'installation et d'exécution. La phase de préparation est celle qui demande le plus d'attention car elle peut comprendre diverses techniques de camouflages, de cryptage et d'inclusion de fichiers distants. Dans le chapitre 3, nous présenterons les différentes techniques utilisées pour la détection des Webshells.

CHAPITRE II.

L'APPRENTISSAGE AUTOMATIQUE

CHAPITRE II.

L'APPRENTISSAGE AUTOMATIQUE*

L'apprentissage automatique est une branche de l'intelligence artificielle (IA) qui se concentre sur le développement et l'utilisation des algorithmes qui apprennent à partir des expériences passées et des données enregistrées. Ces algorithmes améliorent leurs précisions au fil du temps sans être programmées pour le faire [7]. Les algorithmes d'apprentissage automatique sont *entraînés* sur des données disponibles afin de faire des prédictions sur de nouvelles données. Plus l'algorithme est bien entrainé (i.e. ; quantité et qualité des données utilisées en phase d'apprentissage), plus les prédictions deviendront précises. L'objectif principal est de permettre aux machines d'apprendre automatiquement sans l'intervention humaine et d'ajuster ses actions en conséquence.

Dans ce chapitre, nous présenterons le principe de l'apprentissage automatique et ses différents types en se focalisant sur l'apprentissage profond.

2.1. Historique de l'apprentissage automatique

L'apprentissage automatique est initialement fondé sur le principe des réseaux de neurones artificiels. L'histoire des réseaux de neurones remonte à 1943, lorsque le neurophysiologiste Warren McCulloch et le mathématicien Walter Pitts ont publié un article décrivant une analogie entre les relais téléphoniques et la connexion des cellules neuronales du cerveau. Warren McCulloch et Walter Pitts ont décidé alors de créer un circuit électrique qui mime le fonctionnement des neurones biologiques, et c'est ainsi que le réseau de neurones artificiel est né [W4].

En 1958, Frank Rosenblatt a conçu le premier réseau de neurones (appelé *Perceptron*) pour la reconnaissance des formes par les ordinateurs. Un autre exemple extrêmement précoce d'un réseau de neurones est venu en 1959, lorsque Bernard Widrow et Marcian Hoff ont créé deux modèles à l'Université de Stanford. Le premier s'appelait ADELINE et pouvait détecter des modèles binaires (e.g. ; dans un flux de bits, il pourrait prédire ce que serait le bit suivant). Le

* Une grande partie de ce chapitre est en commun avec un autre étudiant en Master de l'année 2021 (Mahboubi AbdelMokim). Cet étudiant est dirigé par Dr Abdelhakim Hannousse et travaille sur l'automatisation d'élaboration des revues systématiques par le biais de l'apprentissage automatique non-supervisé et semi-supervisé. Dans cette version du chapitre on se focalisant sur l'apprentissage profond.

second modèle s'appelait MADELINE et pouvait éliminer l'écho sur les lignes téléphoniques, il avait donc une application utile dans le monde réel. Malgré le succès de MADELINE, il n'y a pas eu beaucoup de progrès dans le domaine jusqu'à la fin des années 1970 pour de nombreuses raisons, principalement la popularité de l'architecture Von Neumann [W4, W5].

Gerald DeJonge en 1981 a introduit le concept d'apprentissage dans lequel un ordinateur analyse les données et crée des règles pour éliminer les informations inutiles [W5]. L'année qui suit a marqué l'intérêt aux développements des réseaux de neurones, lorsque John Hopfield a suggéré de créer un réseau doté de lignes bidirectionnelles, similaires au fonctionnement réel des neurones [W4].

En 1998, les recherches des laboratoires AT&T Bell ont abouti à une bonne précision dans la détection des codes postaux manuscrits du service postal américain. Depuis 2000, de nombreuses entreprises ont réalisé que l'apprentissage automatique augmenterait le potentiel de calcul. C'est pourquoi plus de recherches ont été faites, afin de garder une longueur d'avance sur la concurrence [W4].

2.2. Applications de l'apprentissage automatique

L'apprentissage automatique est particulièrement présent et appliqué dans différents domaines. Les applications classiques incluent la reconnaissance des écritures, de visages et de paroles et la traduction automatiques des textes. Les applications récentes incluent la cyber-sécurité, la recommandation de produits et les véhicules autonomes. Les applications de l'apprentissage automatique sont multiples et peuvent perfectionner sensiblement différents domaines. En fonction de la nature des données, de la masse à traiter et de l'utilisation des informations obtenues, le choix d'appliquer un tel type d'apprentissage automatique va pouvoir varier. Quoi qu'il en soit, l'apprentissage automatique dispose donc d'un véritable potentiel et peut permettre à de nombreux domaines de s'améliorer [7]. Pour une vision plus large sur l'applicabilité de l'apprentissage automatique sur les différents domaines, la figure 2.1 montre l'ensemble des applications où l'apprentissage automatique a marqué son succès depuis son apparence en 1981.

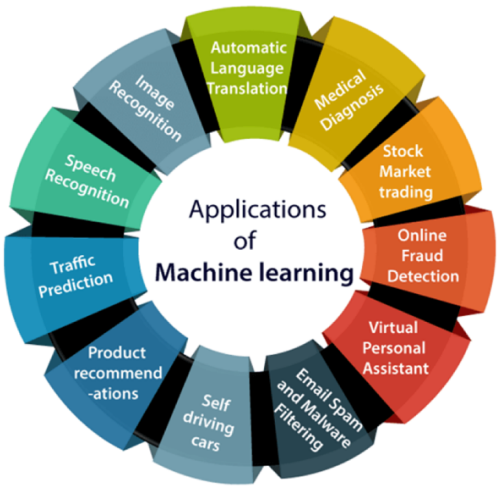

Figure 2.1. Applications concrètes de l'apprentissage automatique

2.3. Le processus d'apprentissage automatique

La conception et le développement d'un modèle d'apprentissage automatique pour résoudre un problème donné est un processus de plusieurs étapes. Le processus implique deux phases fondamentales (voir la figure 2.2): la *préparation des données* et la *conduite et l'évaluation du modèle d'apprentissage*. Chaque phase comporte un ensemble d'étapes élémentaires qui peuvent être exécutées de manière séquentielle ou en parallèle. Ces étapes sont expliquées en détails dans les sous sections qui suivent. L'extraction des caractéristiques est une étape non nécessaire pour l'apprentissage profond (voir la section 2.3.5.3), elle est donc désignée dans la figure 2.2 par un rectangle pointillé.

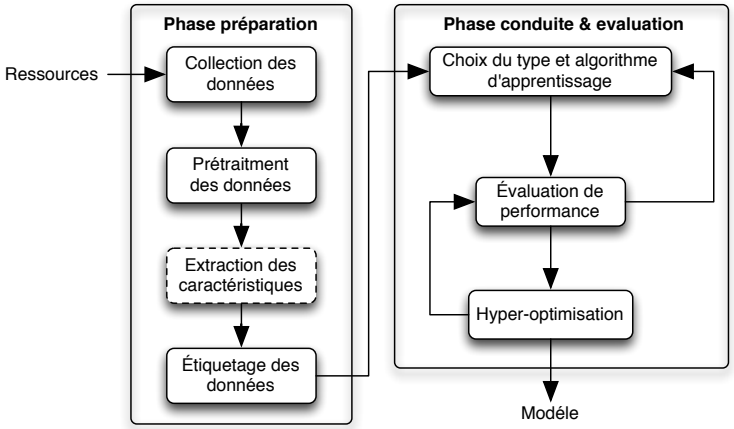

Figure 2.2. Processus général de l'apprentissage automatique

2.3.1. Collection des données

La collecte des données est le fondement du processus d'apprentissage automatique. L'apprentissage sur des données similaires ou des données de types particuliers peuvent rendre le modèle complètement inefficace pour d'autres types de données. C'est pourquoi il est impératif que les considérations nécessaires soient prises lors de la collecte des données, car les erreurs commises à cette étape ne feront que s'amplifier au fur et à mesure pendant la progression vers les dernières étapes [7].

2.3.2. Prétraitement des données

Le prétraitement des données consiste à nettoyer et à préparer les données pour la phase d'apprentissage. Cela comprend le filtrage (e.g.; suppression des bruits), le formatage (e.g.; utiliser le même codage), la normalisation (e.g.; garder la même taille pour tous les individus) et le traitement des données manquantes (e.g. ; récupération des informations manquantes ou élimination des individus avec des informations insuffisantes). Cette étape est la plus délicate, 80% des efforts pour la réalisation des modèles d'apprentissage efficaces est généralement consacrée au prétraitement des données [W6]. Le prétraitement des données est un moyen d'assurer que les données utilisées pour l'apprentissage d'un algorithme sont exactes, complètes et pertinentes. D'un côté, l'utilisation des données incomplètes ou brutes pour l'apprentissage peut entraîner plusieurs erreurs, ce qui entraînera en fin de compte une précision globale beaucoup plus faible. D'un autre côté, des données bien conçues peuvent améliorer l'efficacité du modèle. Il est donc judicieux d'examiner l'ensemble des données utilisées afin qu'elles puissent produire des résultats meilleurs et significatifs.

2.3.3. Extraction des caractéristiques

L'extraction des caractéristiques consiste à transformer des données brutes en un ensemble d'information qui représentent mieux le problème sous-jacent des modèles prédictifs, ce qui améliore la précision du modèle sur des nouvelles données. En d'autres termes, l'extraction des caractéristiques permet de transférer les données collectées en des éléments que l'algorithme d'apprentissage peut comprendre. Les connaissances des domaines sont fréquemment utilisées pour en tirer la meilleure partie. Si cette étape est effectuée correctement, la puissance prédictive des algorithmes d'apprentissage automatique en sera forcément améliorée.

2.3.4. Étiquetage des données

L'étiquetage des données est un élément clé de la préparation des données pour l'apprentissage automatique, car il spécifie les décisions à partir desquelles le modèle apprendra. Une étiquette est le résultat (i.e. ; réponse de modèle) souhaité pour chaque individu de l'ensemble de données utilisé dans la phase d'apprentissage. Bien que certains types d'apprentissage ne nécessitent pas de données étiquetées (voir la section 2.3.5), de nombreux systèmes d'apprentissage automatique s'appuient toujours sur des données étiquetées pour apprendre et exécuter les tâches qui leur sont données.

2.3.5. Choix du type et algorithme d'apprentissage

Les algorithmes utilisés dans l'apprentissage automatique se divisent en trois types de bases (voir Figure 2.3.) : *apprentissage supervisé*, *non-supervisé* et *apprentissage profond*.

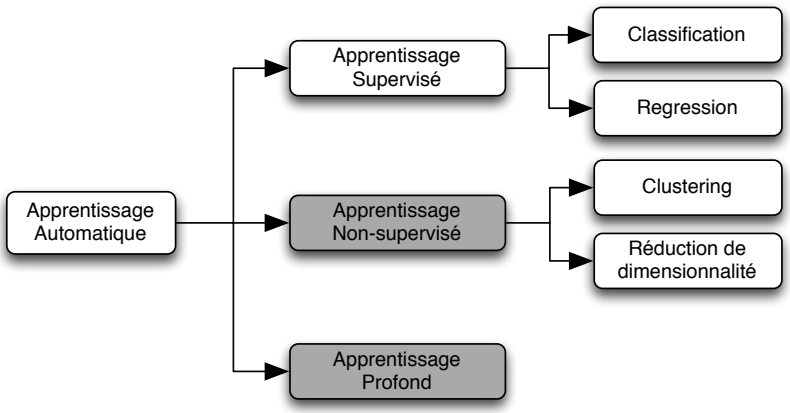

Figure 2.3. Taxonomie des techniques d'apprentissage automatique

2.3.5.1. Apprentissage supervisé :

L'apprentissage supervisé consiste à entrainer les algorithmes exclusivement sur des données étiquetées (i.e.; avec des résultats connues pour chaque individu de l'ensemble de données utilisée en apprentissage) pour réaliser des prédictions sur des nouveaux individus non-étiquetés. Pendant la phase d'apprentissage, l'algorithme supervisé cherche à apprendre les relations entre les données en entrées et les étiquettes qui leur correspondent, afin de déduire des règles d'apprentissage. Ces règles sont par la suite utilisées pour prédire les étiquettes pour un nouveau jeu de données. Le principal avantage de l'apprentissage supervisé est qu'il nous

permet de recueillir des informations ou de produire un rendement d'informations à partir des expériences passées. Deux techniques de prédictions sont utilisées dans l'apprentissage supervisé : la *classification* et la *régression*.

A. Classification :

Les algorithmes de classification prévoient des valeurs discrètes [7]. Ces algorithmes visent à classer les données en entrée en deux ou plusieurs catégories. Les algorithmes de classification couramment utilisés sont : les machines à vecteurs de support (SVM), naïve bayésienne (NB) et les k plus proches voisins (KNN).

B. Régression :

Les algorithmes de régression prévoient des valeurs continues [7]. Les algorithmes à base de régression sont donc utilisés si les réponses à prédire sont des valeurs réelles. Les algorithmes de régression couramment utilisés sont : arbre de décisions (DT), forêt d'arbres décisionnels (RF) et réseaux de neurones (ANN).

2.3.5.2. Apprentissage Non-supervisé :

L'étiquetage des données nécessite généralement l'intervention des experts humains. Dans certains domaines, cette opération peut devenir difficile voire fastidieuse ou impossible lorsque le nombre des données est important. L'apprentissage non supervisé prend en charge le problème en apprenant des données non-étiquetées ce qui réduit le risque d'erreur humaine et la disponibilité des experts.

Le principe de l'apprentissage automatique non-supervisé est de regrouper les individus de l'ensemble de données sans aucune connaissance préalable de ses étiquettes. L'apprentissage non-supervisé alors vise à découvrir les structures sous-jacentes et intrinsèques dans les données d'entrées pour savoir prédire sur des nouvelles données [7]. L'apprentissage non-supervisé est relativement simple et rapide à faire. Les techniques les plus répondues de ce type d'apprentissage sont : la technique de *clustering* et la *réduction de dimensionnalités*.

A. Clustering :

Le clustering est une technique qui divise l'ensemble de données en entrées, en un certain nombre de groupes appelés *clusters* afin que les individus de l'ensemble de données appartenant à un seul cluster aient des caractéristiques similaires. Un cluster n'est rien d'autre qu'un regroupement des individus, la distance entre les individus au sein d'un même cluster

est donc minimale. Les algorithmes couramment utilisés pour le clustering sont: les K-moyennes (K-Means) et le clustering hiérarchique (Hierarchical Clustering).

B. Réduction de dimensionnalité :

La réduction de dimensionnalité est une technique d'apprentissage non-supervisé qui vise à réduire le nombre de caractéristiques utilisées pour la catégorisation des individus de l'ensemble de données. En effet, cela permet d'une part, de réduire l'impact des caractéristiques non pertinentes qui peuvent induire l'algorithme d'apprentissage en erreur. D'autre part, cela permet de limiter le nombre de possibilités à tester, ce qui permet de doubler la vitesse de l'algorithme d'apprentissage. Pour cet effet, les caractéristiques sont généralement combinées afin d'obtenir un plus petit nombre de nouvelles caractéristiques symboliques plus expressives et/ou moins redondantes.

Parmi les algorithmes couramment utilisés pour la réduction des dimensionnalités, on peut citer : l'analyse en composants principales (PCA), l'analyse sémantique latente (LSA) et l'allocation de dirichlet latente (LDA).

2.3.5.3. Apprentissage profond :

L'apprentissage profond est l'une des nouvelles techniques de l'apprentissage automatique. L'apprentissage profond se diffère de l'apprentissage automatique traditionnel par l'absence de l'étape de l'extraction des caractéristiques. Cette dernière est une étape très difficile et très coûteuse en temps [8]. Les modèles d'apprentissage automatique profond sont généralement classés en modèles supervisés et modèles non-supervisés. La figure 2.4 montre quelques modèles de l'apprentissage profond.

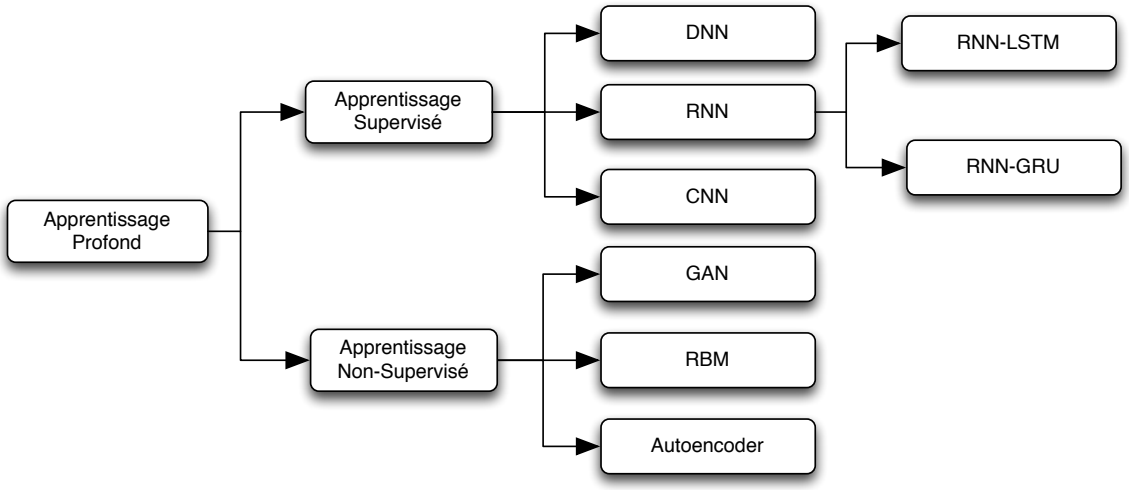

Figure 2.4. Les algorithmes de l'apprentissage profond

Les algorithmes de l'apprentissage profond sont basés sur l'utilisation des réseaux de neurones artificiels (ou Artificiel Neural Networks en anglais) ; ce réseau comporte plusieurs neurones et de nombreux couches cachées hiérarchiques qui constituent une structure complexe. Ceci explique le sens du terme *profond* [9]. Ces neurones permettent de gérer une grande partie des données bruts. À partir de ces derniers et grâce aux multiples transformations non linéaires dans les multiples couches de traitement, ce modèle peut extraire des caractéristiques et d'apprendre à travers chaque couche qui donne un haut niveau d'abstraction [10].

L'augmentation de la quantité des données d'entrainement disponible et l'évolution du matériel informatique, sont deux facteurs importants qui contribuent à l'efficacité des techniques d'apprentissage profond, qui sont utilisées dans divers domaines et secteurs tels que : classification d'image et la reconnaissance vocale [10].

A. Réseau neurones artificiels (ANN) :

Les réseaux de neurones artificiels (ANN) sont des programmes informatiques d'inspiration biologique, conçus en s'inspirant du fonctionnement du cerveau humain, où tous les neurones artificiels sont associés en couches dans le but de transformer les entrées en sortie significatives [11]. Notamment, chaque neurone artificiel est une unité élémentaire, où il reçoit des données d'entrées de neurone précédent, chacun de ces entrées à un poids w qui représente la force de connexion. À travers la fonction d'activation et en utilisant la somme de ces poids, chaque unité va faire sortir les éléments significatifs par son issue unique [12]. La figure 2.5 montre la structure formelle d'un neurone de l'ANN où :

- a_i : sont les entrées de neurone.
- b : chaque neurone possède une entrée supplémentaire de valeur 1, et b est son propre poids de connexion . Cela garantit que même lorsque toutes les entrées sont nulles ($\forall i, a_i = 0$), il y aura toujours une activation dans le neurone.
- w_i : les poids correspondant aux entrées.
- z : la fonction d'intégration des entrées.
- g : fonction d'activation.
- a_{out} : la sortie du neurone.

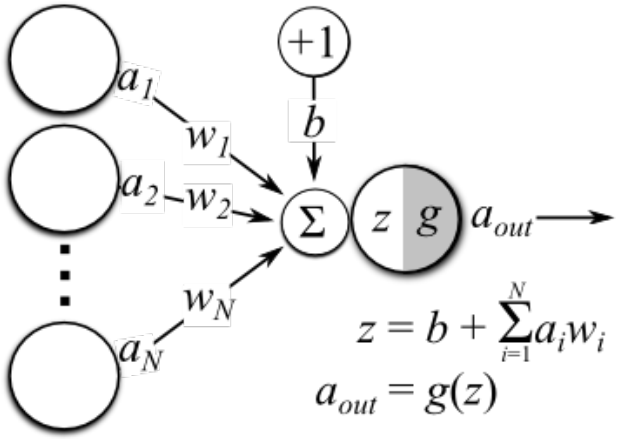

Figure 2.5. La structure formelle d'un neurone [13]

B. Fonction d'activation :

La fonction d'activation est une fonction mathématique qui sert à transformer le signal entrant dans une neurone en signal de sortie ou réponse. Le but de la fonction d'activation est de normaliser la sortie de chaque neurone et d'introduire des complexités non-linéaires au réseau. Le choix de la fonction d'activation a un impact important sur la performance d'un réseau neuronale [14]. La non-linéarité des fonctions d'activation permet de transformer le résultat de la somme pondérée des entrées d'un neurone en une valeur de sortie plus significative au réseau. Cela permet aux réseaux de neurones d'apprendre des données complexes. Dans les fonctions d'activation certains seuils sont utilisés pour déterminer la valeur d'argument de la fonction d'activation qui sera ensuite utiliser pour déterminer l'état de neurone comme suit :

- Somme pondérée >= seuil : l'argument de fonction devient positif ou nul.
- Somme pondérée < seuil : l'argument de fonction devient négatif.

D'après ces cas, les résultats finals de la somme pondérée seront déterminés comme suit :

- En dessous du seuil : le neurone est considéré comme non-actif
- Aux alentours du seuil : le neurone est considéré en phase de transition.
- Au-dessus du seuil : le neurone est considéré comme actif.

Parmi les fonctions d'activation les plus utilisées, nous citons : *Sigmoïde*, *Relu* et *Softmax*.

B.1. Fonction Sigmoïde :

Cette fonction est considérée comme l'une des fonctions d'activations la plus anciennes et la plus utilisée, car elle est introduite de la non linéarité, elle est aussi continuellement différentiable, et simple à dériver [15]. La sortie de la fonction sigmoïde est comprise entre 0 et 1. La fonction sigmoïde définit par :

$$f(x) = \frac{1}{1 + e^{-x}}$$

B.2. Fonction d'unité linéaire rectifiée (Relu) :

La fonction Relu est caractérisée par son efficacité en termes de calcul par rapport à la fonction Sigmoïde. Cela est nécessaire pour l'optimisation des réseaux neurones profonds [15]. La fonction Relu est définie par :

$$f(x) = \max(0, x) = \begin{cases} x, & si\ x \geq 0 \\ 0, & si\ x < 0 \end{cases}$$

B.3. Fonction Softmax :

La fonction Softmax est considérée comme une fonction d'activation de sortie, elle se trouve presque dans les dernières couches des réseaux neuronaux. Softmax calcule la distribution de probabilité à partir d'un vecteur de nombre réels. Cette fonction est utilisée principalement dans les modèles multi-classes où elle renvoie les probabilités de chaque classe, la classe cible ayant la probabilité la plus élevée. Elle est définie par la formule suivante [15].

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

C. Les différentes couches d'un réseau de neurones :

Un réseau de neurones est constitué d'un ensemble de neurones réparties sur plusieurs couches. Chaque couche a une particularité et un rôle dans l'apprentissage du réseau. Dans ce qui suit, nous présenterons les différentes couches les plus utilisées pour l'apprentissage profond :

C.1. La couche Dense :

C'est la couche la plus simple et la plus utilisé pour l'apprentissage profond. La couche Dense connecte ses neurones à tous les neurones de la couche précédente et elle prend en compte

toutes les connexions possibles. En pratique, la couche Dense doit être initialisé en indiquant le nombre des neurones dans la couche et la fonction d'activation qui doit être attribuée à chacun de ses neurones [16]. Si la couche Dense est utilisée comme une couche d'entrée de réseau, il faut aussi indiquer la forme de données d'entrée (i.e.; dimension).

C.2. La couche Dropout :

Entraîner un grand réseau de neurones sur des ensembles de données relativement petits peuvent surcharger les données d'apprentissage. Cela peut réduire la performance des modèles lorsqu'ils sont évalués sur de nouvelles données. La couche dropout, appelée aussi couche d'exclusion, a pour but de réduire l'effet de ce phénomène en éliminant quelques neurones où les met à 0 et réduit le calcul dans le processus d'apprentissage. La couche dropout est toujours ajoutée après une couche Dense dans un réseau d'apprentissage profond. Une des paramètres essentiels de cette couche est la fraction des neurones à éliminer (valeur entre 0 et 1) [16].

C.3. La couche d'intégration (Embedding) :

C'est une couche d'entrée pour les réseaux de neurones destinés à apprendre des données textuelles. Après le codage des données textuelles en entrées par des entiers, de sorte que chaque mot soit représenté par un entier positif unique, la couche d'intégration transforme ces entiers positifs en vecteurs de taille fixe. La couche intégration est initialisée avec des poids aléatoires et apprendra une intégration pour tous les mots de l'ensemble de données d'apprentissage. C'est une couche flexible qui peut être utilisée de deux manières :

1. Elle peut être utilisée où l'intégration est apprise avec le modèle lui-même.
2. Elle peut être utilisée pour charger un modèle d'intégration de mots pré-entraîné.

Trois arguments de base doivent être indiqués lors de l'initialisation d'une couche d'intégration [16] : taille du vocabulaire dans les données textuelles, la taille de l'espace vectoriel dans lequel les mots seront intégrés (i.e.; la taille des vecteurs de sortie de cette couche) et la longueur des séquences d'entrée.

C.4. La couche de convolution :

Il existe plusieurs types de couches de convolution: Conv1D, Conv2D, SeparableConv1D, DepthwiseConv2D et Conv3DTranspose [16]. Chacune de ces couches ayant sa propre fonctionnalité et utilisé selon la nature des données d'apprentissage. Par exemple, la couche Conv1D est généralement utilisée pour la classification des textes, alors que la couche Conv2D est destinée à la classification d'images. Une couche de convolution a pour but d'appliquer

certains filtres aux données en entrées pour repérer la présence d'un ensemble de caractéristiques utiles. Les paramètres les plus utilisés pour initialiser une couche de convolution sont : le nombre de filtres (caractéristiques repérées) dans la convolution, la longueur de la fenêtre de convolution et la fonction d'activation utilisée. Lorsqu'une couche de convolution est utilisée comme une couche d'entrée, il faut aussi fournir la forme de données d'entrée.

C.5. La couche pooling :

Ce type de couche est souvent placé entre deux couches de convolution : elle reçoit en entrée plusieurs caractéristiques et sert à réduire ce nombre, tout en préservant les caractéristiques importantes. Il existe différents types de couches pooling [16]. Par exemple, la couche MaxPooling1D sous-échantillonne l'ensemble de caractéristiques d'entrée en prenant la valeur maximale sur une fenêtre spatiale d'une taille spécifique utilisée pour l'initialisation de cette couche.

C.6. La couche LSTM :

La couche LSTM c'est l'une des couches les plus importante dans les réseaux d'apprentissage profond. Elle permet d'inclure une cellule mémoire capable de maintenir les informations en mémoire pendant de longues périodes [17]. Les caractéristiques d'entrée sont séparées en données historiques et prédictives. Les données historiques sont entrées dans la couche LSTM pour modéliser les relations entre les données observées dans le passé [18].

C.7. La couche GRU :

Cette couche est un développement étendu de la couche LSTM. La couche GRU atteint des performances comparables à celles de LSTM, mais elle utilise moins de paramètres, ce qui la rend plus rapide à former [19].

D. La fonction de perte :

La fonction de perte ou (*loss function* en anglais), c'est une fonction qui calcule l'écart quadratique moyen entre la valeur prédite et la valeur réelle attendue dans les réseaux de neurones. Cette valeur évalue la performance des réseaux. Les réseaux qui ont la valeur de *perte* la plus faible sont les réseaux les plus performants [9].

E. Les différentes architectures des réseaux d'apprentissage profond :

Dans ce qui suit, nous présenterons les différentes architectures les plus utilisées pour l'apprentissage profond :

E.1. Le réseau neuronal profond (DNN) :

Le DNN représente l'architecture standard qui a permis d'atteindre toutes les autres architectures connues aujourd'hui. Le DNN est un réseau entièrement connecté, qui se compose principalement de plusieurs couches Dense, comme il a été montré dans la figure 2.6. L'ensemble de données entrant au réseau sont introduits dans la couche Dense d'entrée, et les entrées du neurone de la première couche cachée sont données par la formule suivante [20]:

$$z_{1(i)} = \sum_{j} W_{1(i,j)} x_j + b_{1(i)}$$

Où : $W_{1(i,j)}$ représentent les poids des connections avec le neurone i, et $b_{1(i)}$ représente le biais associé au neurone i.

La sortie du neurone i dans la première couche cachée est donnée par :

$$a_{1(i)} = ActivationFunction(z_{1(i)})$$

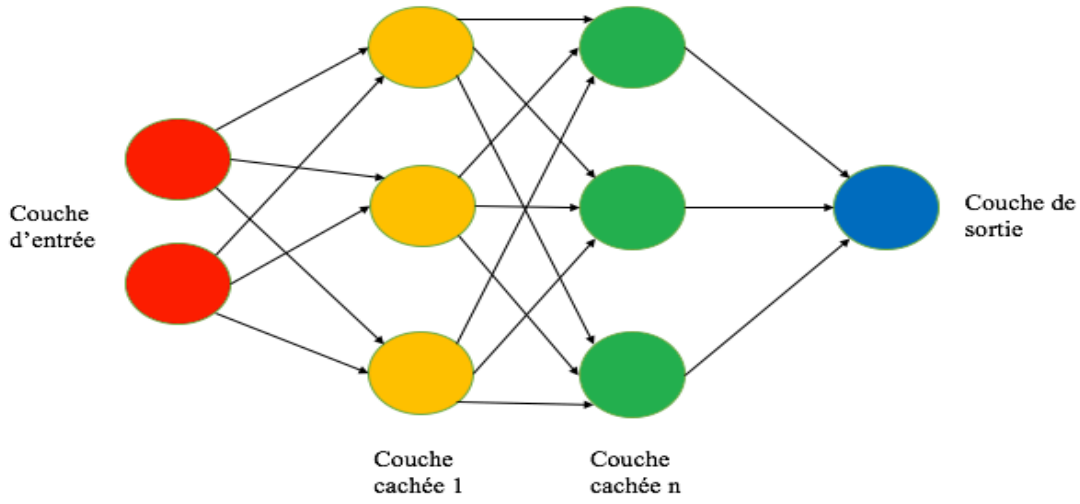

Figure 2.6. L'architecture d'un réseau DNN

E.2. Le réseau de neurones récurrents (RNN) :

Dans un réseau DNN, l'indépendance des entrées et des sorties les unes des autres représente un inconvénient majeur, car lorsqu'on veut prédire des prochaines entrées, il faut connaitre les entrées précédentes. L'idée d'utiliser les réseaux neurones récurrents est venue pour traiter ce problème [8]. Les RNNs, sont des réseaux qui contiennent des boucles, et qui permettent aux

informations de persister. Les RNNs utilisent des informations séquentielles. Le terme récurrent signifie l'exécution de la même tâche pour chaque élément d'une séquence, la sortie étant dépendante aux calculs précédents. En d'autres termes, les RNNs ont une mémoire qui capture l'information sur ce qui a été déjà calculé. Pour cela, les RNNs utilisent généralement soit un ou plusieurs couches LSTM soit un ou plusieurs couches GRU ce qui nous donne les architectures : RNN-LSTM et RNN-GRU.

E.3. Le réseau de neurones convolutif (CNN) :

L'émergence des réseaux de neurones convolutifs a permis aux ordinateurs de réaliser l'une des capacités fondamentales de l'être humain, qui est l'analyse des objets de son environnement, qui passe par la reconnaissance des éléments de notre champ de vision. Les CNNs, sont un type de réseau de neurone spécialisés pour le traitement de données ayant une topologie semblable à une grille, qui peuvent être considérées comme une grille 1D (vecteur) et une grille 2D de pixels (matrice). Les CNN utilisant au moins une couche de convolution dans leurs couches [16].

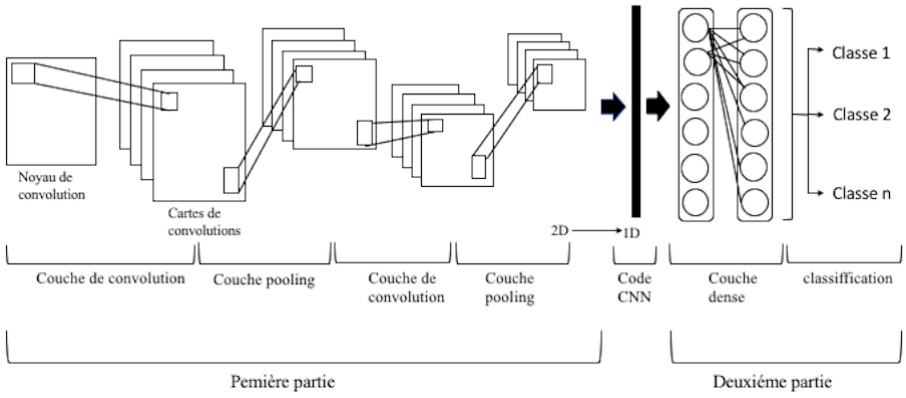

Figure 2.7. L'architecture d'un réseau CNN

2.3.6. Évaluation de performance

Quelque soit le type d'apprentissage utilisé (supervisé, non-supervisé ou profond), après la phase d'apprentissage, un modèle sera créé. Il est nécessaire de vérifier le bon fonctionnement et la généralisation de ce modèle. L'évaluation de la prédiction d'un modèle avec les mêmes données qui ont été utilisées pour l'apprentissage n'est pas utile. Pour évaluer correctement un modèle, il doit être testé sur des données qui ne faisaient pas partie des données d'apprentissage. Les résultats de prédiction doivent être comparés aux valeurs des résultats connues.

2.3.6.1. Méthodes de validation

Pour valider correctement les modèles d'apprentissage, deux méthodes de validation sont utilisées : *échantillonnage* (Sampling) et la *validation croisée* (cross-validation).

A. Échantillonnage (Sampling) :

L'échantillonnage consiste à diviser l'ensemble collecté de données en deux parties : une partie pour l'apprentissage et l'autre pour le test. Différentes techniques d'échantillonnages sont utilisées selon la nature et la taille de l'ensemble de données : *aléatoire*, *rejet* et *préférentielle*. La figure 2.8 montre un exemple simple d'un échantillonnage où l'ensemble de données de 16 individus est divisé en deux parties égales ; une pour l'apprentissage (8 individus) et une autre pour le test (8 individus).

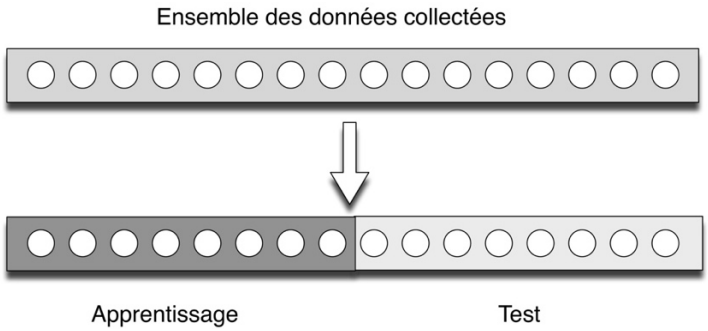

Figure 2.8. Exemple sur l'échantillonnage

B. Validation croisée (cross-validation) :

La cross-validation consiste simplement à diviser l'ensemble de données collectée en k échantillons. Un des k échantillons est sélectionné pour validation alors que les k-1 autres échantillons sont utilisés pour l'apprentissage. Le processus de validation se répète k fois, en sélectionnant à chaque fois un échantillon différent pour la validation. En effet, à chaque fois, un modèle différent est généré et sa performance est mesurée est sauvegardée. La moyenne et l'écart type des k scores de performances peuvent être calculés pour estimer le biais et la variance de la performance de validation [21]. La figure 2.9 montre un exemple sur la validation croisée à 5 parties. L'ensemble de données est divisé en 5 parties égales. Les individus pour la partie du test sont désignés par des jetons blancs alors que les individus des parties d'apprentissage sont désignés par des jetons noirs. Dans le cas où le nombre des individus n'est pas divisible par le nombre des parties souhaité, la dernière partie doit inclure le reste de la division.

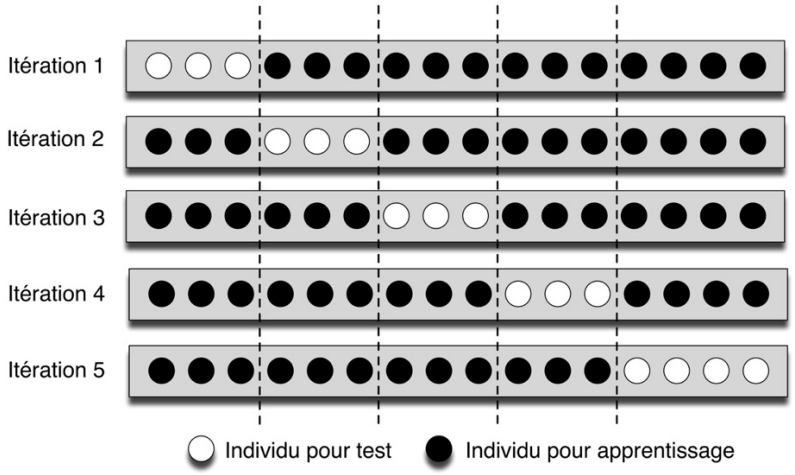

Figure 2.9. Exemple sur la validation croisée

2.3.6.2. Mesures de performance

Les mesures de performance sont des métriques récapitulatives indiquant la qualité de correspondance entre les valeurs prévues et les valeurs obtenues par le modèle. Toutes ces mesures sont basées sur la matrice de confusion décrites dans le tableau 2.1 pour une classification binaire :

		Prédiction	
		C1	**C2**
Valeurs réelles	**C1**	Vrai positif (VP)	Faux négatif (FN)
	C2	Faux positif (FP)	Vrai négatif (VN)

Tableau 2.1. Matrice de confusion.

D'après la matrice de confusion, le vrai positif (VP) indique le nombre des individus de l'ensemble de validation qui sont correctement classés en C1, contrairement au faux négatif (FN) qui indique le nombre des individus de C1 qui sont mal classés en C2. Un vrai négatif (VN) montre le nombre des individus qui sont correctement classés en C2 alors que le faux positif (FP) indique le nombre des individus de C2 qui sont mal classés en C1. Voici la liste des mesures généralement adoptées pour la comparaison et validation des modèles :

A. Rappel : C'est la proportion des individus de la classe C_i qui ont été effectivement identifiées par le modèle.

$$Rappel_{C_i} = \frac{VP_{C_i}}{VP_{C_i} + FN_{C_i}}$$

B. Précision : C'est la proportion des individus de C_i qui ont été effectivement identifiés correctement par le modèle.

$$Precision_{C_i} = \frac{VP_{C_i}}{VP_{C_i} + FP_{C_i}}$$

C. F1-score : C'est la moyenne harmonique entre la précision et le rappel. Par conséquent, ce score prend en compte à la fois les faux positifs et les faux négatifs. F1-Score est essentiellement utile surtout si la répartition des classes est inégale.

$$F1 - score_{C_i} = 2 \times \frac{Precision_{C_i} \times Rappel_{C_i}}{Precision_{C_i} + Rappel_{C_i}}$$

2.3.7. Hyper-optimisation

Chaque algorithme d'apprentissage automatique comporte un ou plusieurs paramètres. Ces paramètres contrôlent la précision du modèle. Par conséquent, les valeurs de ces hyper-paramètres sont particulièrement importantes pour améliorer la performance des modèles. L'hyper-optimisation est le processus de sélection d'hyper-paramètres optimaux pour le modèle conçu. Il est souvent recommandé d'ajuster ces hyper-paramètres selon la nature du problème étudié et l'ensemble de données utilisé pour l'apprentissage. Deux techniques de bases sont utilisées pour ce faite : la *recherche de grille* et la *recherche aléatoire*.

2.3.7.1. La recherche de grille :

La recherche de grille fonctionne en essayant chaque combinaison possible de valeurs de paramètres que nous voulons essayer pour un modèle. La recherche de grille est effectuée de manière automatique mais peut devenir coûteuse en termes de calcul si le nombre des valeurs à explorées est important.

2.3.7.2. Recherche aléatoire :

La méthode de recherche aléatoire consiste à utiliser des valeurs des hyper-paramètres sélectionnées de manière aléatoires pour obtenir la meilleure solution pour un modèle. L'inconvénient de la recherche aléatoire est qu'elle peut dans certains cas manquer de valeurs significatives dans l'espace de recherche.

2.4. Conclusion :

Pour conclure, l'apprentissage automatique se développe rapidement dans le domaine de l'intelligence artificielle. L'apprentissage automatique a monté son efficacité pour résoudre des problèmes trop complexes pratiquement dans tous les domaines évolutifs. Spécifiquement, l'efficacité de l'apprentissage profond a été approuvée dans différents domaines. Pour les systèmes complexes, différentes techniques d'apprentissage automatique peuvent être combinés pour avoir une meilleure performance. L'efficacité de l'apprentissage automatique indique que dans les prochaines années, ce domaine de recherche sera un élément principal dans tous les domaines de notre vie quotidienne, sans oublier, que la capacité des algorithmes d'apprentissage profonds augmente progressivement et se rapproche de plus en plus de la capacité de l'être humain. Sachant qu'il existe certains domaines où la capacité de ces algorithmes dépasse la capacité humaine tels que : domaine de classification des images, domaine de reconnaissance de visages. Dans les prochains chapitres, nous expérimenterons et nous proposerons un système basé sur l'apprentissage automatique pour la détection efficace des Webshells pour assurer la sécurité des serveurs Web.

CHAPITRE III.

DETECTION DES WEBSHELLS PAR L'APPRENTISSAGE AUTOMATIQUE : ÉTAT DE L'ART ET ANALYSE DE L'EXISTANT

CHAPITRE III.

DETECTION DES WEBSHELLS PAR L'APPRENTISSAGE AUTOMATIQUE : ÉTAT DE L'ART ET ANALYSE DE L'EXISTANT

Vu l'augmentation du nombre des attaques Webshells signalés ces dernières années (voir Chapitre 1), les chercheurs essayent de trouver des solutions dans le but de proposer de nouvels outils et d'améliorer l'efficacité des systèmes de protection contre ce genre d'attaques. Plusieurs méthodes et techniques ont été proposées, utilisées et examinées. Une étude récente menée par Hannousse et Yahiouche [22] couvre l'ensemble des articles scientifiques parus ces dernières années sur la détection et la prévention des Webshells. L'étude de Hannousse et Yahiouche [22] montre que 71% des solutions proposées à ce jour utilisent l'apprentissage automatique, seules 29% ont proposées d'autres types de solutions comme techniques d'isolation et diversifications pour la prévention de ces attaques.

L'apprentissage automatique est devenu donc l'une des technologies importantes et populaires utilisées pour la détection des différents malwares notamment les Webshells [22]. Différents algorithmes de classification (classifieurs) ont été utilisés. Ces classifieurs doivent être d'abord entraînés sur des échantillons validés pour prédire et classer les nouveaux échantillons (voir Chapitre 2). Vu le nombre des Webshell détectés et déjà reconnus, l'apprentissage supervisé est devenu un moyen utile pour la détection des Webshells. Différents modes d'apprentissage automatique supervisés ont été utilisés dans la littérature [22] : *apprentissage simple*, *apprentissage ensembliste*, *apprentissage profond* et *apprentissage mixte*. La figure 3.1 montre le pourcentage d'utilisation de chaque mode. Dans ce chapitre, nous expliquerons brièvement chaque mode d'apprentissage et nous présenterons en détails quelques études de chaque mode.

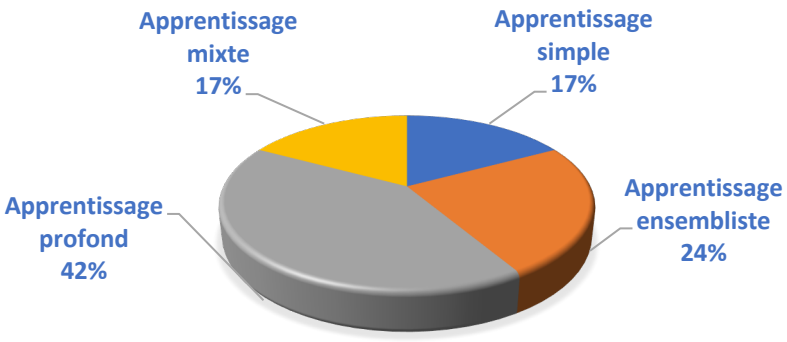

Figure 3.1. Distribution des solutions proposées pour la détection des Webshells [22]

3.1. Détection par apprentissage simple

L'apprentissage simple implique l'utilisation des classifieurs primitifs pour le développement des modèles de prédiction. Par conséquent, des classifieurs statistiques, probabilistes et de régressions ont été utilisés individuellement pour la détection automatique des Webshells.

Sun et collab. [1] ont proposé un nouvel algorithme supervisé pour identifier les Webshells non connus. Un ensemble de caractéristiques extraites des codes sources des Webshells connus sont énumérées et utilisées pour construire une matrice de notation. Cette matrice est utilisée par la suite pour une phase d'analyse afin d'obtenir des poids associés à chaque caractéristique et construire un modèle de prédiction des nouveaux Webshells [2]. Le tableau 3.1 ci-dessous présente les différentes caractéristiques utilisées dans cet algorithme :

Caractéristiques textuelles	Autres Caractéristiques
Nombre de mots	Appels de fonction de manipulation de caractères
Nombre de mots différents	Appels de fonction clé
Longueur maximale des mots	Appels de fonction de chiffrement et déchiffrement
Longueur totale du texte	Appels de fonction système
Nombre de notes	Appels de fichier
Nombre de caractère spécial	Appels de contrôle activeX
	Appels de base de données
	Nombre de script

Tableau 3.1. L'ensemble des caractéristiques utilisées par Sun et collab. [1]

Guo et collab. [4] ont adopté l'utilisation des séquences des codes d'opérations (opcodes) extraites des codes sources des Webshells. Ces séquences sont obtenues après une compilation des codes sources et correspond aux jeux d'instructions d'un langage assembleur. Les séquences extraites sont encodées (i.e. ; *vectorisées*) pour être utilisées dans la construction d'un modèle d'apprentissage sur la base d'un classifieur probabiliste : Naïve Bayésienne (NB). L'encodage a été réalisée en utilisant une méthode de pondération nommée TF-IDF (*term frequency-inverse document frequency*). Les expériences menées dans [4] ont montré que le modèle proposé surpasse d'autres classifieurs tels que la Machine à vecteurs de support (SVM) [22].

3.2. Détection par apprentissage ensembliste

Dans un apprentissage ensembliste, deux ou plusieurs classifieurs sont combinés pour obtenir une meilleure prédiction que celle obtenue avec des classifieurs simples pris individuellement.

Huang et collab. [23] ont expérimenté deux types de caractéristiques : statistiques et abstraites. Les caractéristiques statistiques comprennent : la taille du mot le plus long, entropie de Shannon et nombre d'utilisation de 21 fonctions et variables à haut risque. Concernant les caractéristiques abstraites, les auteurs ont examiné les séquences opcodes des sources compilées encodées par TF-IDF. Les auteurs ont constaté qu'un modèle constitué d'une Forêt d'arbres décisionnelles (RF) est plus performant que d'autre classifieurs comme : les k-plus proches voisins (KNN) et la Machine à vecteurs de support (SVM). La particularité de cette étude est l'utilisation de deux types de séquences d'opcodes : celle qui correspond aux chemins d'exécution dans le code, et celle qui correspond au code lui-même. Ces caractéristiques ont été combinées pour former une matrice de valeurs utilisée pour l'apprentissage et le test des classifieurs. Le modèle ensembliste à base de RF a montré des performances améliorées par rapport aux autres modèles [22].

Fang et collab. [24] ont proposé un algorithme d'apprentissage ensembliste qui combine deux classifieurs : FastText et RF. Initialement, le modèle FastText est d'abord entraîné sur des séquences d'opcodes générées à partir des scripts PHP. Le résultat de l'entrainement (la prédiction de FastText) est ensuite combiné avec 5 caractéristiques statistiques : la taille du mot le plus long, entropie de Shannon, indice de coïncidence, appels aux fonctions dangereuses et présence de certains mots-clés dans les commentaires des sources. Ces

caractéristiques sont mesurées pour chaque Webshell et combinées avec la prédiction de FastText et l'opcode encodé. La matrice résultante de la combinaison est enfin utilisée pour l'apprentissage et le test par un classifieur ensembliste : RF.

3.3. Détection par apprentissage profond

L'apprentissage profond est devenu une technique largement adoptée pour la détection des malwares. Selon l'étude menée par Hannousse et Yahiouche [22], 42% des études existantes à ce jour utilisent l'apprentissage profond pour la détection des Webshells. Ces statistiques montrent la popularité de ce mode d'apprentissage dans le domaine de détection des Webshells. L'apprentissage profond est basé essentiellement sur l'utilisation des réseaux neuronaux artificiels, ce que nous avons déjà expliqué et détaillé au deuxième chapitre. Dans ce qui suit, nous présentons les deux modèles performants dans ce contexte.

Tao et collab. [25] ont expérimenté trois modèles : Perceptron multicouche (MLP), Réseau neuronal convolutif (CNN) et Mémoire à long court terme (LSTM). Les auteurs ont constaté que les modèles CNN et LSTM convergent mieux en facteur de temps que le modèle MLP. Sauf que ce dernier surpasse les autres modèles en termes de prédiction. L'entrainement et les tests de ces modèles ont été appliquées sur des codes sources des fichiers PHP. Ces derniers sont encodés en utilisant la technique nommée *Sac de Mots* (BoW).

Lv et collab. [26] ont expérimenté différents modèles CNN, chacun de ces modèles a été entrainé et testé sur une différente technique d'encodage des codes sources des Webshells : *Encodage 1 parmi n* (One-hot), *Sac de Mots* (BoW) et *Mot au Vecteur* (Word2Vec). Le modèle CNN avec l'encodage BoW a donné le meilleur score.

3.4. Détection par apprentissage mixte

Le mode d'apprentissage mixte implique deux ou plusieurs classifieurs de différents modes d'apprentissage montrés ci-dessus : apprentissage simple, apprentissage ensembliste, et/ou apprentissage profond.

Ai et collab. [27] ont proposé un modèle d'apprentissage mixte qui met en œuvre une combinaison d'un pool de 3 classifieurs de base : Régression logistique (LR), Forêt d'arbres décisionnels (RF) et un Perceptron multicouche (MLP). Par l'utilisation de l'algorithme de programmation séquentielle des moindres carrés (SLSQP), une valeur de poids est calculée

pour chaque classifieur de base. Ensuite, ces valeurs sont utilisées pour prédire les nouveaux échantillons. Le modèle a utilisé une combinaison de 5 caractéristiques statistiques (taille du mot le plus long, entropie de Shannon, indice de coïncidence, taux de compression de fichier, appels aux fonctions dangereuses) et des opcodes encodés par Word2Vec. Un algorithme génétique est utilisé pour réduire la dimension des vecteurs de caractéristiques.

Yong et collab. [28] ont proposé deux modèles pour la protection des réseaux IoT (Internet des Objets). Un modèle ensembliste simple (M1) à base de forêt d'arbre décisionnels (RF) pour les dispositifs disposant de ressources modérées (mémoire, processeur/contrôleur). Un autre modèle ensembliste complexe (M2) basé sur le vote de 6 classifieurs pour les appareils qui ont des capacités informatiques plus puissantes. Les classifieurs de bases utilisées pour le deuxième modèle sont : k-plus proches voisins (KNN), Naïve bayésienne (NB), Arbre de décision (DT), K-moyenne (K-Means), Machine à vecteurs de support (SVM) et un Perceptron multicouche (MLP). Avec l'utilisation de TF-IDF pour encoder les séquences d'opcodes, des vecteurs de caractéristiques de taille 100 ont été générés. Ces vecteurs sont utilisés pour former les deux modèles proposés [22].

Pour résumer, le tableau 3.2 ci-dessous indique les méthodes de validation adoptées, le détail des ensembles de données utilisées et les meilleures performances obtenus par les auteurs des travaux cités auparavant.

	#Normal	**#Webshell**	**Méthode de validation**	**Performance (F1-score %)**
Modèles à base d'apprentissage simple				
Sun et collab. [1]	1,002	347	Échantillonnage aléatoire 8:2	98.20
Guo et collab. [4]	1,002	500	Échantillonnage aléatoire 7:3	97.00
Modèles à base d'apprentissage ensembliste				
Huang et collab. [1]	11,397	912	Échantillonnage aléatoire 7:3	98.40
Fang et collab. [4]	6,934	1,587	Validation-croisée à10 parties	97.79
Modèles à base d'apprentissage profond				
Tao et collab. [25]	-	-	Échantillonnage aléatoire 8:2	99.50
Lv et collab. [26]	8,361	3,944	Validation-croisée à 10 parties	99.30

Modèles à base d'apprentissage mixte				
Ai et collab. [25]	5,379	571	Validation-croisée à 7 parties	98.90
Yong et collab. [26]	2,593	1,551	Échantillonnage aléatoire 8:2	M1 : 98.32 M2 : 97,92

Tableau 3.2. Validation et performance des modèles d'apprentissage pour la détection automatique des Webshells [22].

3.5. Conclusion

Dans ce chapitre, nous avons présenté les différentes solutions récentes proposées pour la détection des Webshells qui se basent sur l'apprentissage automatique. Ainsi, nous avons décrit un aperçu sur quelques travaux réalisés dans chaque type de solution avec une explication détaillée sur chaque modèle utilisé. Cela nous a permis de déterminer les failles et l'ensemble des modèles les plus performants. Spécifiquement, le tableau 3.2 montre l'efficacité des modèles d'apprentissage profond par rapports aux autres modèles. Malheureusement, malgré la diversité des solutions proposées aucun de ces travaux n'a publié l'intégralité de l'ensemble des données utilisées pour l'apprentissage des modèles élaborés. Cela empêche une comparaison objective des différentes solutions proposées. Sur la base de l'étude menée par Hannousse et Yahiouche [22], nous allons proposer, dans le prochain chapitre, une étude qui consiste à expérimenter deux modèles d'apprentissage profond et leurs combinaisons avec un modèle d'apprentissage ensembliste (modèle d'apprentissage mixte). Un ensemble de données unique est utilisé pour valider les différents modèles.

CHAPITRE IV.

UN MODELE D'APPRENTISSAGE PROFOND POUR LA DETECTION DES WEBSHELLS EN PHP

CHAPITRE IV.

UN MODELE D'APPRENTISSAGE PROFOND POUR LA DETECTION DES WEBSHELLS EN PHP

Suivant l'étude réalisée dans le chapitre précédent, et sur la base des résultats obtenus par les études récentes, on a constaté que les modèles d'apprentissage profond sont les modèles les plus performants pour la détection des webshells malveillants. Pour cette raison, nous proposons et nous examinons, dans ce chapitre, un modèle d'apprentissage profond DNN (Deep Neural Network). Nous comparons la performance de notre modèle avec d'autres modèles concurrents.

4.1. Modèle proposé

Parmi les nombreux modèles d'apprentissage profond, nous proposons une architecture DNN constituée de nombreuses couches entièrement connectées. Ce choix est pris pour deux raisons essentielles :

1. Une architecture rarement utilisée dans le contexte de notre projet (i.e.; détection des webshells)
2. Un modèle rapide et conforme avec les contraintes matérielles disponibles. Certains autres architectures comme CNN et LSTM ont besoin d'un processus d'apprentissage très long et des machines performantes. Bien sûre que des plateformes spécifiques comme *googlecolab* peuvent aussi être utilisées pour pallier ce problème, mais les CNNs et LSTMs sont déjà utilisées pour la détection des webshells (Voir chapitre III).

Le modèle proposé est constitué de plusieurs couches : entrée, cachés et de sortie :

4.1.1. Couche d'entrée

Comme entrée du réseau, nous avons adopté deux couches :

1. Une couche entièrement connectée *Dense* avec 512 nœuds et une fonction d'activation '*relu*'. Cette fonction nous a donné le meilleur résultat par rapport aux autres fonctions d'activations.
2. Une couche *Dropout* avec une valeur égale à 0.5. Cela permet de désactiver temporairement 50% des neurones dans la couche *Dense* ainsi que toutes ses connexions entrantes et sortantes. Cette approche est couramment utilisée pour réduire l'*overfitting* lors de l'entraînement des modèles.

4.1.2. Couches cachées

Nous utilisons 6 couches cachées identiques constituées du même nombre des nœuds 512 et la même fonction d'activation 'relu'. Chaque couche cachée est constituée d'une couche entièrement connectée *Dense* suivie d'une couche de régularisation *Dropout*.

4.1.3. Couche de sortie

Pour la couche de sortie, nous utilisons une couche entièrement connectée *Dense* avec seulement deux nœuds (pour modéliser une classification binaire), et une fonction d'activation '*softmax*'.

Pour la configuration du modèle, nous utilisons :

1. La fonction de perte *'sparse_categorical_crossentropy*'. Cette fonction est utilisée pour évaluer la capacité d'apprentissage du modèle en termes des poids associés à chaque connexion entre deux neurones dans le réseau.
2. L'optimiseur '*Adam*' pour mettre à jour les poids de réseau afin de réduire la valeur de la perte avec le minimum d'erreur.

La figure 4.1 montre l'architecture globale du modèle proposé.

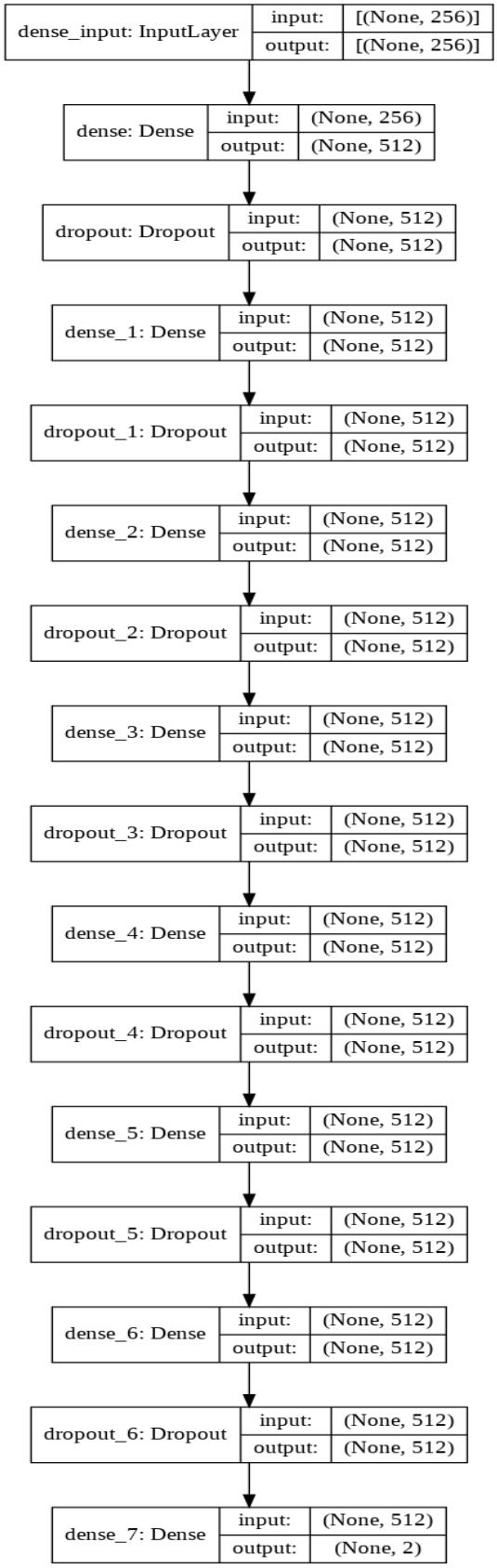

Figure 4.1. Architecture du modèle proposé

4.2. Processus de validation

Dans cette section nous décrivons le processus général adopté pour la validation de notre modèle. La figure 4.2 montre les différentes étapes de ce processus.

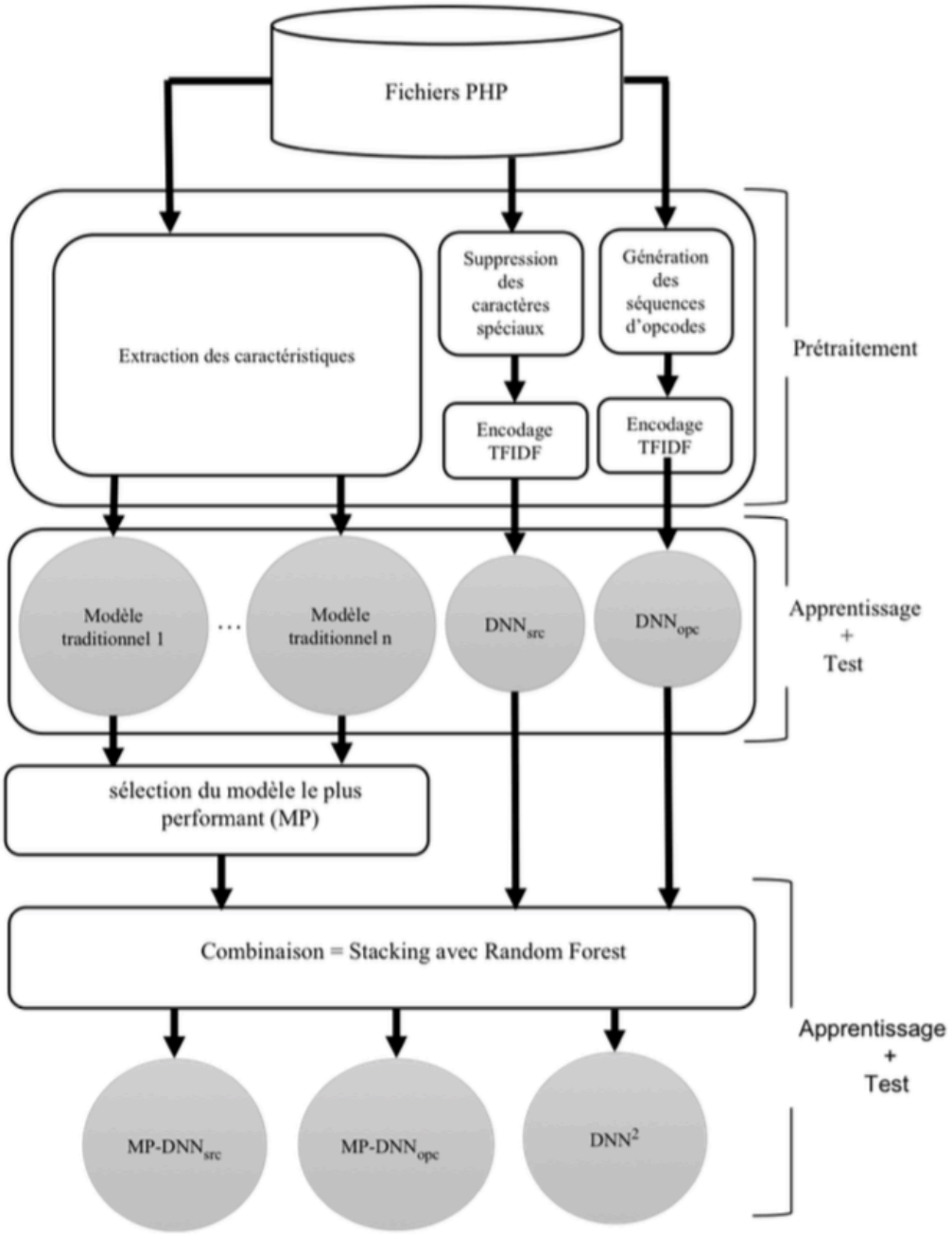

Figure 4.2. Processus de validation

Le processus décrit ci-dessus est constitué de trois étapes fondamentales : Collection de données, Prétraitement et Validation (Apprentissage + Test). Dans ce qui suit, nous détaillons chacune de ces étapes.

4.2.1. Collection de données

L'ensemble de données utilisé dans note étude est le résultat d'efforts de deux étudiants : Djaghout Yahia et Grini Achraf dans leur projet de fin d'étude License réalisé au département d'informatique de l'Université 8 Mai 1945 Guelma en Juillet 2021 [29]. L'ensemble de données collectés en [29] inclus des webshells écrits en différents langages de scripts : PHP, ASP, ASPX, JSP et Perl. Nous utilisons ces écrits en PHP. L'ensemble de données choisi est constitué de 1984 fichiers PHP, 992 fichiers bénins, et 992 fichiers webshells. Notons que la qualité et la quantité de l'ensemble de données utilisé dans l'apprentissages des modèles jouent un rôle très important dans la détermination de la performance de ces modèles. Afin d'assurer une bonne qualité des données, un processus de filtrage a été adopté dans [29] dans le but d'éviter l'inclusion des dupliques dans l'ensemble de données. Le processus de filtrage consiste à appliquer une fonction de hachage cryptographique (MD5 pour Message Digest 5) pour obtenir l'empreinte numérique de chaque fichier. Deux fichiers sont identiques si leurs empreintes numériques sont égales. Dans le cas des fichiers PHP, la fonction MD5 a été aussi appliquée aux séquences d'opcodes générées de chaque fichier. Cela permet d'identifier des scripts ayant les mêmes séquences d'opérations élémentaires et non seulement des codes sources identiques. Cela permet d'assurer un haut degré de diversité entre les fichiers PHP inclus dans l'ensemble de données.

4.2.2. Prétraitement des données

La phase de prétraitement, dans notre projet, consiste à transformer un code brut (i.e. ; un script PHP) en formats utiles pour la phase d'apprentissage et le test des modèles. Revenant vers la figure 4.2, nous distinguons 4 opérations distinctes de prétraitements. Dans ce qui suit, nous expliquons chacune de ces opérations avec un exemple démonstratif.

4.2.2.1. Suppression des caractères spéciaux :

La suppression des caractères spéciaux est nécessaire pour pouvoir identifier uniquement les mots significatifs dans le script. Cela est réalisé par l'utilisation des expressions régulières. Dans notre projet, nous avons adopté l'expression suivante pour ne garder que les séquences alphanumériques :

[^a-zA-Z0-9]+

À titre démonstratif, en appliquant l'expression précédente sur un script de webshell simple : <?PHP eval($_POST ["cmd"]);?>, nous obtenons la séquence des mots suivants :

PHP eval POST cmd

4.2.2.2. Génération des séquences d'opcodes :

L'opcode est l'abréviation de **Op**eration **Code**, c'est-à-dire les codes d'opérations qui spécifient la séquence des jeux instructions après la compilation des scripts PHP. Lorsqu'un script PHP est exécuté, la machine virtuelle PHP va transformer le script lu en un langage intermédiaire qui représente des opérations de base. Ces instructions de bas niveau sont généralement appelées *opcodes*. Plusieurs techniques peuvent être utilisées pour visualiser l'opcode d'un fichier PHP. Une solution simple consiste à installer une extension PHP qui nous permettra de visualiser la représentation interne des scripts exécutés. Cette extension est nommée VLD (Vulcan Logic Disassembler). La figure 4.3 montre le résultat de l'application du VLD sur un fichier contenant le code d'un webshell simple : <?PHP eval($_POST ["cmd"]);?>

```
L1    #0    FETCH_R<2>            "_POST"                          ~0
L1    #1    FETCH_DIM_R           ~0                 "cmd"         ~1
L1    #2    INCLUDE_OR_EVAL<1>    ~1
L1    #3    RETURN<-1>            1
```

Figure 4.3. Application du VLD sur un fichier PHP

Le résultat obtenu en appliquant VLD doit ensuite être traiter afin d'extraire la séquence des opérations qui se trouvent sur la troisième colonne. Pour cela, nous utilisons l'expression régulière suivante qui détermine la séquence des lettres majuscules et le caractère de soulignement :

\s(\b[A-Z_]+\b)\s

En appliquant cette expression sur l'exemple montré dans la figure 4.3, nous obtenons la séquence d'opcodes suivante :

FETCH_R FETCH_DIM_R INCLUDE_OR_EVAL RETURN.

4.2.2.3. Encodage TFIDF :

Le but de l'étape d'encodage est de passer de la représentation écrite des mots à leurs représentations vectorielles numériques par l'utilisation de plusieurs techniques. Parmi lesquelles, nous choisissons la technique TFIDF.

TFIDF ou (*Term Frequency–Inverse Document Frequency*) mesure le pouvoir discriminant d'un mot ou d'un groupe de mots dans un document donné. Essentiellement, elle mesure l'importance d'un certains mots dans un document par rapport aux autres documents dans une collection. La formule mathématique de cette technique est définie par :

$$TFIDF(t) = TF(t) \times \log\left(\frac{N}{DF(t)}\right)$$

Avec :

TF : représente le nombre d'occurrences du terme *t* dans le document analysé.

N : représente le nombre total de documents dans la collection.

DF : représente le nombre de documents dans lequel le terme *t* est présent

Cette mesure est utilisée pour pondérer les termes-clés : c'est-à-dire, plus la valeur TFIDF d'un terme est élevée, plus celui-ci est important dans le document analysé en prenant compte de tous les documents dans la collection. A la fin de l'étape de l'encodage, nous obtenons une matrice indiquons l'importance de chaque terme dans les différents documents de la collection.

À titre démonstratif, imaginons la liste des scripts PHP suivante :

```
<?php eval($_POST["cmd"]);?>
<?php base64_decode('ZXZhbCAoX1BPU1QgWyJQIl0pOw==');?>
<?php system("system");?>
```

Après la suppression des caractères spéciaux de chaque script, nous obtenons les séquences des termes suivants :

```
'php eval POST cmd'
'php base64 decode ZXZhbCAoX1BPU1QgWyJQIl0pOw'
'php system system'
```

En appliquant la fonction TFIDF à ces séquences, nous obtenons la matrice suivante :

$$\begin{bmatrix} 0. & 1. & 0. \\ 0.861037 & 0.50854232 & 0. \\ 0. & 0.28321692 & 0.95905588 \end{bmatrix}$$

Chaque ligne représente un document et chaque colonne représente les valeurs TFIDF d'un mot (caractéristique) dans chaque document. Pour simplicité, dans cet exemple, nous avons fixé le nombre de caractéristiques à 3. Les 3 caractéristiques les plus importantes choisies par TFIDF sont respectivement : 'base64', 'php', et 'system'. Le mot 'php' (voir colonne 2) se présente dans tous les documents, il a donc une valeur d'importance dans chaque document. Les deux autres mots : 'base64' et 'system' ne se figure que dans un seul document, ils possèdent donc des valeurs nulles dans les documents où les mots n'apparaissent pas.

4.2.2.4. Extraction des caractéristiques des webshells :

La grande similarité entre les webshells malveillants et les scripts normaux augmente la difficulté de pouvoir les distinguer. Les expériences et les études élaborées dans ce domaine permettent l'identification d'un ensemble de caractéristiques de webshells. Ces caractéristiques peuvent être classées en fonction de leurs sources d'extraction [22]. Dans ce projet, nous nous concentrons sur les caractéristiques extraites des scripts webshells (100) qui peuvent être classées selon leur nature en 3 classes distinctes :

1. *Caractéristiques statistiques* : comparent certaines valeurs statistiques des fichiers webshells avec des fichiers normaux pour identifier la présence des codes cryptés. Nous distinguons sept caractéristiques de base :

   C1. *Indice de coïncidence* [8] : permet de mesurer le taux ou la fréquence de répétition des caractères dans un texte en utilisant la formule suivante :

   $$IC = \frac{\sum_{i=1}^{c} n_i(n_i-1)}{\frac{N(N-1)}{c}}$$

   Où N représente la longueur du texte, n_i représente la fréquence de chacune des lettres de l'alphabet i et c représente le nombre total des caractères.

   C2. *Entropie* [8] : quantifie la valeur attendue des informations contenues dans un texte ou un fichier, généralement en unités telles que les bits. La formule ci-dessous calcule l'entropie d'un texte et renvoie le nombre de bits par caractère nécessaires pour représenter les données contenues dans le texte. Plus ce nombre est élevé, plus l'entropie est présente dans la chaîne de données, ce qui indique un degré élevé d'aléas ou de variété des informations.

$$H = -\sum_{i=1}^{N} x_i \times log_2 x_i$$

Où x_i représente un mot et N représente la longueur du texte.

C3. *Taille du mot le plus long* : identifie la longueur de la plus longue chaîne ininterrompue dans un fichier. Ce calcul nous aide à identifier l'utilisation des codes cryptés dans un fichier.

C4. Nombre d'expressions méchantes : calcule le nombre d'occurrence des expressions méchantes dans un fichier PHP.

C5. Nombre d'expressions super méchantes : calcule le nombre d'occurrence des expressions super méchantes dans un fichier PHP. Les expressions super méchantes sont : *'@$_[]=', '$_=@$_GET', '$_[+ ""]='*.

C6. *Utilisation de la fonction eval avec des variables en paramètres :* compte le nombre d'appels à la fonction *eval* avec des variables en paramètres dans un fichier PHP.

C7. *Rapport de compression* : compare la taille d'un fichier avec sa taille après compression.

2. *Caractéristiques lexicales :* indiquent la manière dont les sources des scripts sont écrites. Les attaquants utilisent plusieurs techniques pendant la création des webshells pour dissimuler leurs objectifs malveillants, comme l'utilisation des mots-clés spécifiques cachés dans des balises ou des commentaires. Cinq caractéristiques de cette catégorie sont considérées :

C8. Taille du fichier

C9. Nombre de mots

C10. Nombre de mots différents

C11. Nombre des caractères spéciaux

C12. Longueur maximale des lignes

3. *Caractéristiques syntaxiques* : se réfèrent aux expressions, variables, et fonctions utilisées dans les scripts PHP. Les attaquants peuvent accéder au système dans le but d'élever les privilèges, et de télécharger des fichiers importants par l'utilisation des fonctions dangereuses. Ils peuvent aussi adapter leurs webshells à la plateforme du serveur ciblé par l'utilisation des boucles et des branchements conditionnelles. Deux caractéristiques de cette catégorie sont considérées :

C13. *Nombre des instructions conditionnelles*

C14. *Nombre de boucles*

C15-58. *Nombre d'appels aux fonctions dangereuses*

C59-75. *Nombre d'utilisation des commandes*

C76-84. *Nombre d'utilisation des variables dangereuses*

C85-100. *Nombre d'occurrence des mots-clés suspects en commentaires*

Le tableau 4.1 résume l'ensemble des caractéristiques considérées dans notre étude :

Type	Index	Description	Détails
Statistique	C1	Indice de coïncidence	
	C2	Entropie	
	C3	Taille du plus long mot	
	C4	Expressions méchantes	
	C5	Expressions super méchantes	

	C6	Eval avec des variables en paramètres	
	C7	Rapport de compression	
Lexicale	C8	Taille totale	
	C9	Nombre de mots	
	C10	Nombre de mots différents	
	C11	Nombre des caractères spéciaux	
	C12	Ligne de longueur maximale	
Syntaxique	C13	Nombre des instructions conditionnelles	‘if’, ‘else’, ‘case’
	C14	Nombre de boucles	‘for’, ‘foreach’, ‘while’
	C15-58	Nombre des fonctions dangereuses	‘base64_encode', 'base64_decode', 'eval', 'str_replace', 'assert', 'system', 'cmd_shell', 'fopen', 'fwrite', 'pcntl 'c99_buff_prepare', 'shell_exec', 'shell', 'exec', 'curl_exec', 'proc_open', 'python_eval', 'file_get_contents', 'curl', 'popen', 'include', 'require', 'include_once', 'array_map', ‘array_walk', 'posix_getpwuid', 'fileowner', ’filegroup', 'posix_getgrgid', 'str_rot13', 'gzencode', 'gzdeflat', 'gzcompress', 'passthru', 'unserialize', 'xpath_eval', 'get_headers', 'get_browser', ‘fgets', 'dlob', 'readdir', 'mysql_fetch_array', 'mysql_fetch_object', 'c99sh_surl'.
	C59-75	Nombre des commandes dangereuses	'wget', 'lynx', 'get', 'fetch', 'perl', 'python', 'gcc', 'chmod', 'nohup', 'nc', 'uname', 'id', 'ver', 'sysctl', 'whoami', 'pwd', '$ostype'.
	C76-84	Nombre des variables dangereuses	'$_get', '$_post', '$_cookie', '$_request', '$_files', '$_session', '$cmd', '$_server', '$_env'

	C85-100	Nombre des mots-clés malicieux en commentaires	'webshell by', 'web shell by', 'hack by', 'bypass AV', 'developed by', 'password is', 'r57', 'c99', 'n3shell', 'tryang team', 'c99shell', 'cod3rz', 'xakep', 'http://ccteam.ru/update/-c999shell', 'http://ccteam.ru/files/c999sh-_sources', 'phpspy'.

Tableau 4.1. Liste des caractéristiques utilisées

4.2.3. Validation

Le processus de validation adopté est constitué de 3 étapes (Voir la figure 4.2) :

1. **Phase 1 :** nous analysons la performance des différents modèles traditionnels simples et ensemblistes. Ces modèles sont entrainés sur l'ensemble des caractéristiques décrites dans le tableau 4.1.
2. **Phase 2 :** nous expérimentons deux versions du modèle proposé : DNNsrc et DNNopc. Les deux modèles ayant la même architecture décrite en section 4.1. La différence réside dans les entrées utilisées pour l'apprentissage et le test de chaque modèle. Spécifiquement, le modèle DNNsrc est entrainé sur l'encodage TFIDF des code sources des fichiers en entrées, alors que le DNNopc est entrainé sur l'encodage TFIDF des séquences d'opcodes générées de chaque fichier en entrée.
3. **Phase 3 :** nous examinons la performance de la combinaison des modèles. Dans cette phase, le meilleur modèle traditionnel trouvé depuis la **Phase 1** est combiné séparément avec les deux versions de notre modèle proposé (DNNsrc, DNNopc) en utilisant la technique d'empilement (*stacking),* où les prédictions des deux modèles de base sont utilisées comme entrées à un autre modèle pour la prédiction finale. Dans cette étude, nous utilisons le classifieur RF (forêt d'arbres décisionnels) pour cette combinaison. Nous examinons aussi la combinaison des deux versions de notre modèle en utilisant aussi la technique *stacking* basée sur RF ; ce modèle est noté par DNN^2.

Tous les modèles examinés sont entrainés et testés sur les mêmes échantillons de l'ensemble de données décrit en section 4.2.1. Nous adoptons la technique de validation croisée d'ordre 10 (Voir chapitre II). Cela nous permet la détection du meilleur modèle avec un haut degré de confiance. Les mesures de performance utilisées sont : Rappel, Précision et F1-score.

4.3. Résultats de validation

Dans cette section, nous présentons les différents résultats obtenus de chaque phase de validation.

4.3.1. Performances des modèles traditionnels

Le tableau 4.2 montre la performance des différents modèles d'apprentissage traditionnels. Le modèle à base de RF donne la meilleure valeur de performance en termes de F1-score 95.69%. Ce dernier sera donc utilisé par la suite dans la phase 3.

Modèle	Recall (%)	Precision (%)	F1 score (%)
RF	95.63	95.76	95.69
Dicision Tree	93.61	93.66	93.61
SVM	31.10	93.74	46.60
KNeighbors	80.30	92.91	86.05
SGD	89.47	70.97	76.82
Gaussin Naive Bayes	42.06	93.80	58.02
Bernouli Naive Bayes	65.07	95.41	77.32
MLP	89.77	84.57	86.66
Logistic Regression	89.24	86.91	87.97

Tableau 4.2. Performance des modèles traditionnels

4.3.2. Performances du modèle proposé

Le but de notre étude est de développer un modèle efficace pour la détection des webshells malveillants. Cela est possible par la minimisation du taux d'erreurs et la maximisation des exactitudes possibles. Plusieurs essais ont été faits afin d'obtenir les bons paramètres des deux versions de notre modèle : DNNsrc et le DNNopc. Ces paramètres ont un grand impact sur les performances des modèles d'apprentissage profond, notamment durant la phase d'apprentissage. Le nombre d'époques (epochs) utilisé dans chaque modèle est considéré comme l'un des paramètres les plus important. Nous avons examiné les deux versions du modèle pendant la phase d'apprentissage. Les figures 4.4 et 4.5 montrent les résultats des tests élaborés. Après l'analyse des résultats obtenus, nous avons fixé le nombre d'epochs à 31 pour DNNsrc et 364 pour DNNopc. Dans le cas de DNNsrc, le modèle se stabilise après 31 itérations où le taux de perte ne dépasse pas 0.03. De même, le modèle DNNopc se stabilise après 364 itérations où la valeur de perte ne dépasse pas 0.003.

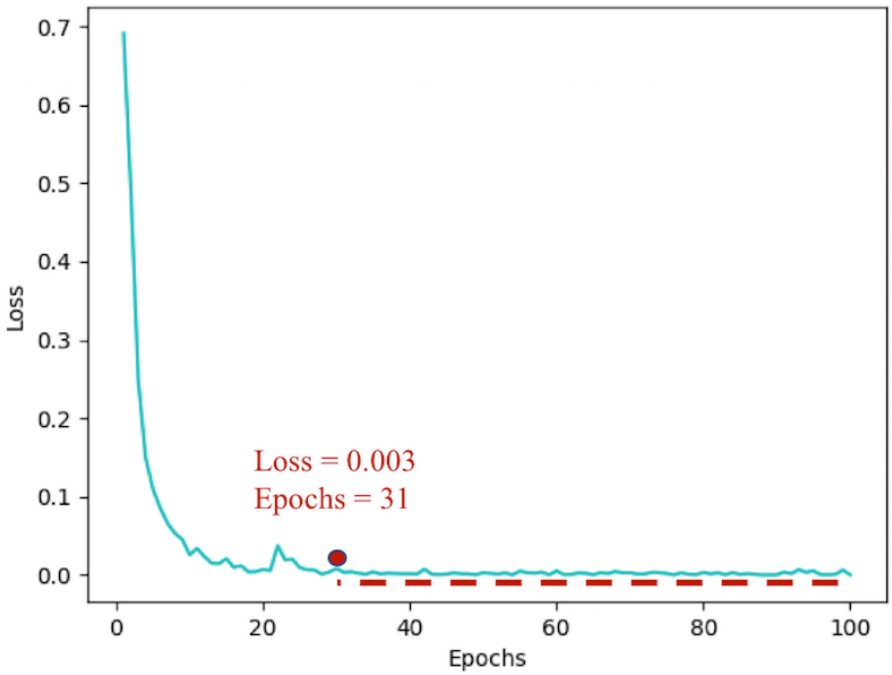

Figure 4.4. Perte vs nombre d'epochs (cas de DNNsrc)

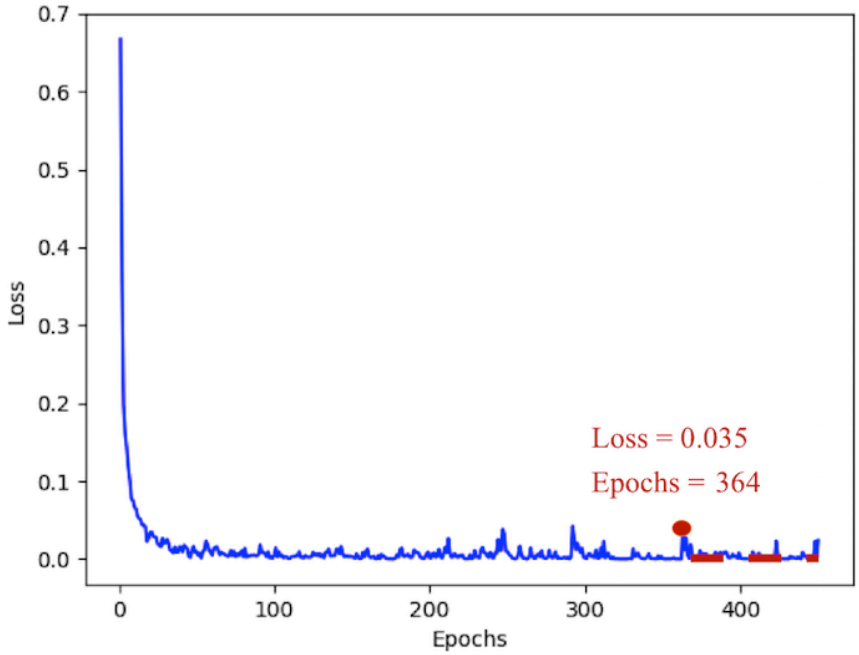

Figure 4.5. Perte vs nombre d'epochs (cas de DNNopc)

Le tableau 4.3 montre la performance des deux versions de notre modèle. La version DNNopc est la plus performante par rapport à DNNsrc (96.79% > 96.22%) et les deux sont plus performants que les modèles traditionnels (comparaison avec les résultats du tableau 4.2).

Modèle	Recall (%)	Precision (%)	F1 score (%)
DNN_{opc}	96.48	97.12	96.79
DNN_{src}	96.59	95.90	96.22

Tableau 4.3. Performance du modèle proposé

4.3.3. Performances des modèles combinés

En combinant RF avec les deux versions DNNsrc et DNNopc et en examinant la combinaison DNN^2 = $Stacking_{RF}$(DNNsrc, DNNopc) nous obtenons les résultats décrits dans le tableau 4.4. D'après les valeurs obtenues, aucune amélioration n'a été détectée et les combinaisons RF-DNNsrc et DNN^2 ont la même performance.

Modèle	Recall (%)	Precision (%)	F1 score (%)
RF-DNN$_{opc}$	96.38	97.02	96.68
RF-DNN$_{src}$	96.59	95.90	96.22
DNN^2	96.59	95.90	96.22

Tableau 4.4. Performance des modèles combinés

4.4. Conclusion

Dans ce chapitre, nous avons examiné plusieurs modèles d'apprentissage automatique pour la détection des webshells écrits en PHP. Le but a été d'obtenir un modèle plus performant. Nous avons proposé un modèle à base d'apprentissage profond. Deux versions de ce modèle ont été évaluées et comparées avec différents autres modèles traditionnels. Les résultats obtenus nous ont convaincus que le modèle DNN entrainé sur des séquences d'opcodes convient à la distinction automatique des webshells malveillants et qu'il est particulièrement efficace. Le modèle donne une meilleure performance comparée aux autres modèles traditionnels.

CHAPITRE V.

PHP WEBSHELL SCANNER : UN OUTIL DE DETECTION AUTOMATIQUE DES WEBSHELLS EN PHP

CHAPITRE V.

PHP WEBSHELL SCANNER : UN OUTIL DE DETECTION AUTOMATIQUE DES WEBSHELLS EN PHP

Afin de mettre en pratique notre approche proposée pour la détection des Webshell, détaillée en chapitre IV, nous développons un outil nommé : PHP Webshell Scanner. Ce dernier permet la détection automatique des Webshells écrits en PHP en examinant un répertoire de fichiers spécifié par l'utilisateur. Dans ce chapitre, nous allons montrer en détail la conception de cet outil et nous allons décrire, effectivement, la démarche suivie et les différents outils utilisés pour l'implémentation de notre outil.

5.1. Conception de l'outil

Pour la conception de notre outil, nous avons utilisé la méthodologie UML (Unified Modeling Language), qui est un langage de modélisation visuel. Cette méthodologie nous a permis de concevoir et de mettre en œuvre le logiciel de manière consistante et facile. Cette modélisation est basée sur les descriptions de différents diagrammes en adoptant certains ensembles de règles. Chaque diagramme décrit une vue différente détaillée du logiciel. La méthodologie UML est considérée comme l'une des méthodes les plus utilisées pour la conception des systèmes complexes [31]. Dans ce qui suit, nous décrivons deux types de diagrammes utilisés pour la conception de notre outil.

5.1.1. Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est le premier schéma du modèle UML, c'est un moyen qui nous a permis d'identifier les caractéristiques fonctionnelles de notre application. En d'autres termes, le diagramme de cas d'utilisation est une description de ce qui se passe lorsque l'utilisateur se connecte à l'application.

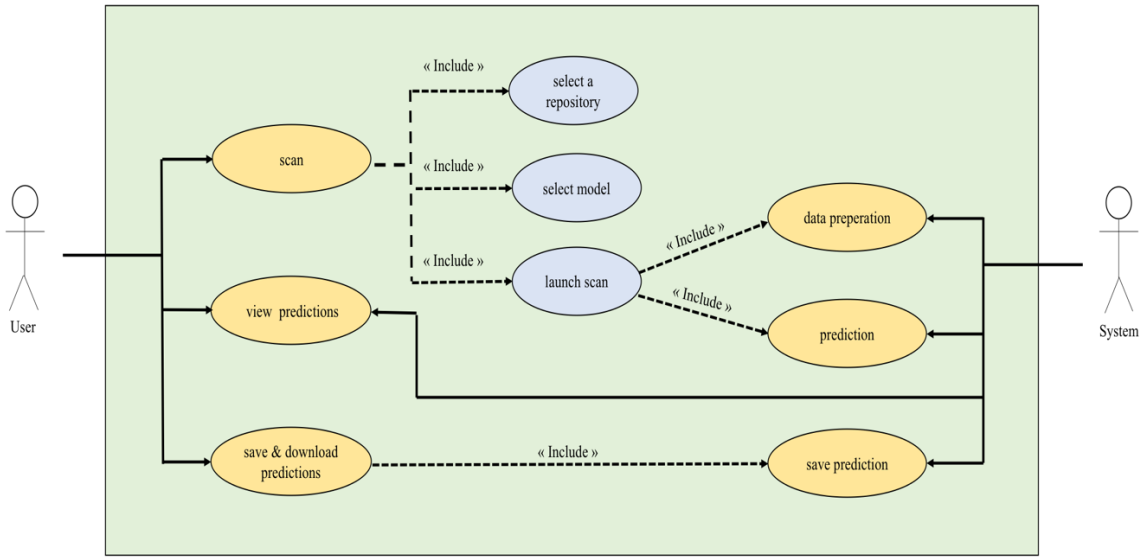

Figure 5.1. Diagramme de cas d'utilisation.

Selon le diagramme de cas d'utilisation décrit dans la figure 5.1, lorsque l'utilisateur accède à notre application, il peut analyser des fichiers pour détecter les fichiers Webshells des fichiers normaux. Pour cela, il doit d'abord sélectionner un répertoire où se trouve les fichiers à analyser et choisir le modèle qu'il veut utiliser pour la détection. À titre académique et comparatif, nous avons intégré dans cet outil 6 modèles d'apprentissage automatiques pour la détection des Webshells (Voir Chapitre IV pour plus de détails) : un modèle traditionnel ensembliste (RF), deux modèles d'apprentissage profond (DNN_{src} et DNN_{opc}) et trois combinaisons des modèles ou modèles hybrides (RF-DNN_{src}, RF-DNN_{opc} et DNN^2). À la fin de la phase d'analyse, l'utilisateur à la possibilité de sauvegarder et télécharger les résultats de la prédiction du système choisi. Cela permet à l'utilisateur de faire des comparaisons avec d'autre modèles concurrents. Le système de notre application intervient pour la préparation de données, la prédiction, affichage et sauvegarde des résultats à la demande.

5.1.2. Diagrammes de séquences

Le diagramme de séquence est le diagramme d'interaction le plus fréquent. Ce diagramme est conçu en fonction du temps, indiquant les opérations à réaliser, les

messages à envoyer et les tâches à accomplir. Pour simplifier la complexité des diagrammes de séquences utilisées dans notre application, nous distinguons six diagrammes de séquences, un pour chaque modèle d'apprentissage intégré dans l'outil. Nous allons discuter dans ce qui suit chacun de ces modèles.

5.1.2.1. Cas d'un modèle traditionnel :

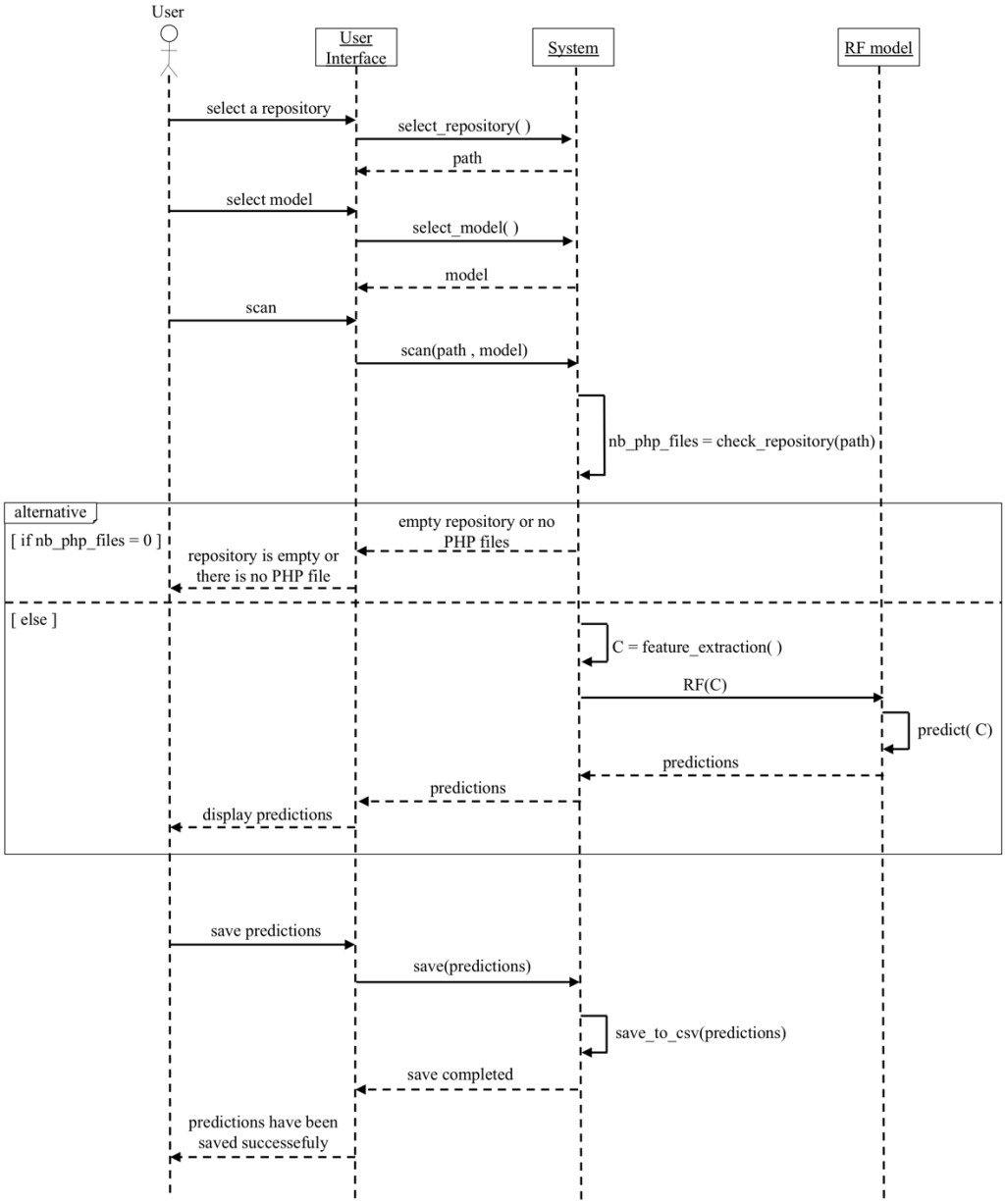

Figure 5.2. Diagramme de séquence : cas d'un modèle traditionnel - RF.

Depuis le diagramme montré dans la figure 5.2, lorsque l'utilisateur sélectionne le modèle RF pour l'analyse des fichiers d'un répertoire donné, le système vérifie d'abord si le répertoire n'est pas vide et qu'il comporte des fichiers PHP. Dans le cas où le répertoire ne contient pas des fichiers PHP, il renvoie un message à l'utilisateur. Sinon, le système va extraire les caractéristiques de chaque fichier PHP dans le répertoire, qui seront envoyés au modèle RF pour être analysés. Une fois l'analyse terminée, le modèle renvoie une prédiction indiquant l'état de chaque fichier (malveillant ou normal). Ces états sont affichés à l'utilisateur en utilisant une interface utilisateur adéquate. Les prédictions peuvent aussi être téléchargés par l'utilisateur en cas de besoin.

5.1.2.2. Cas d'un modèle d'apprentissage profond :

Dans le cas où l'utilisateur choisit l'un des modèles d'apprentissage profond, DNN_{src} ou le DNN_{opc}, le système va suivre la même démarche de vérification selon que le répertoire choisi par l'utilisateur comporte des fichiers PHP ou pas. Si le répertoire contient au moins un fichier PHP, le système va interagir selon le modèle d'apprentissage choisi : DNN_{src} ou DNN_{opc}. Les diagrammes décrits dans les figures 5.3 et 5.4 montrent en détail les séquences d'opérations et interactions en cas de sélection des modèles DNN_{src} et DNN_{opc} respectivement.

Dans le cas de DNN_{src}, le système va récupérer le contenu de chaque fichier et va le nettoyer en supprimant tous les caractères spéciaux, puis il envoie le texte nettoyé au modèle $TFIDF_{src}$. Ce dernier va construire le vecteur de caractéristiques de chaque fichier. Le vecteur construit par $TFIDF_{src}$ est par la suite envoyé au modèle DNN_{src} pour être analysé. Le système affichera alors à l'utilisateur les résultats de l'analyse avec la possibilité de sauvegarder et télécharger ces résultats pour une utilisation ultérieure.

Dans le cas de DNN_{opc}, le système va générer les séquences d'opcodes de chaque fichier PHP dans le répertoire. Ces opcodes seront envoyés au modèle $TFIDF_{opc}$ dans le but de construire les vecteurs de caractéristiques de chaque fichier. Ces vecteurs seront renvoyés au système, qui les envoie au modèle DNN_{opc} pour la phase d'analyse. Une fois l'analyse terminée, le modèle DNN_{opc} envoie ses prédictions au système, qui les affiche à son tour à l'utilisateur.

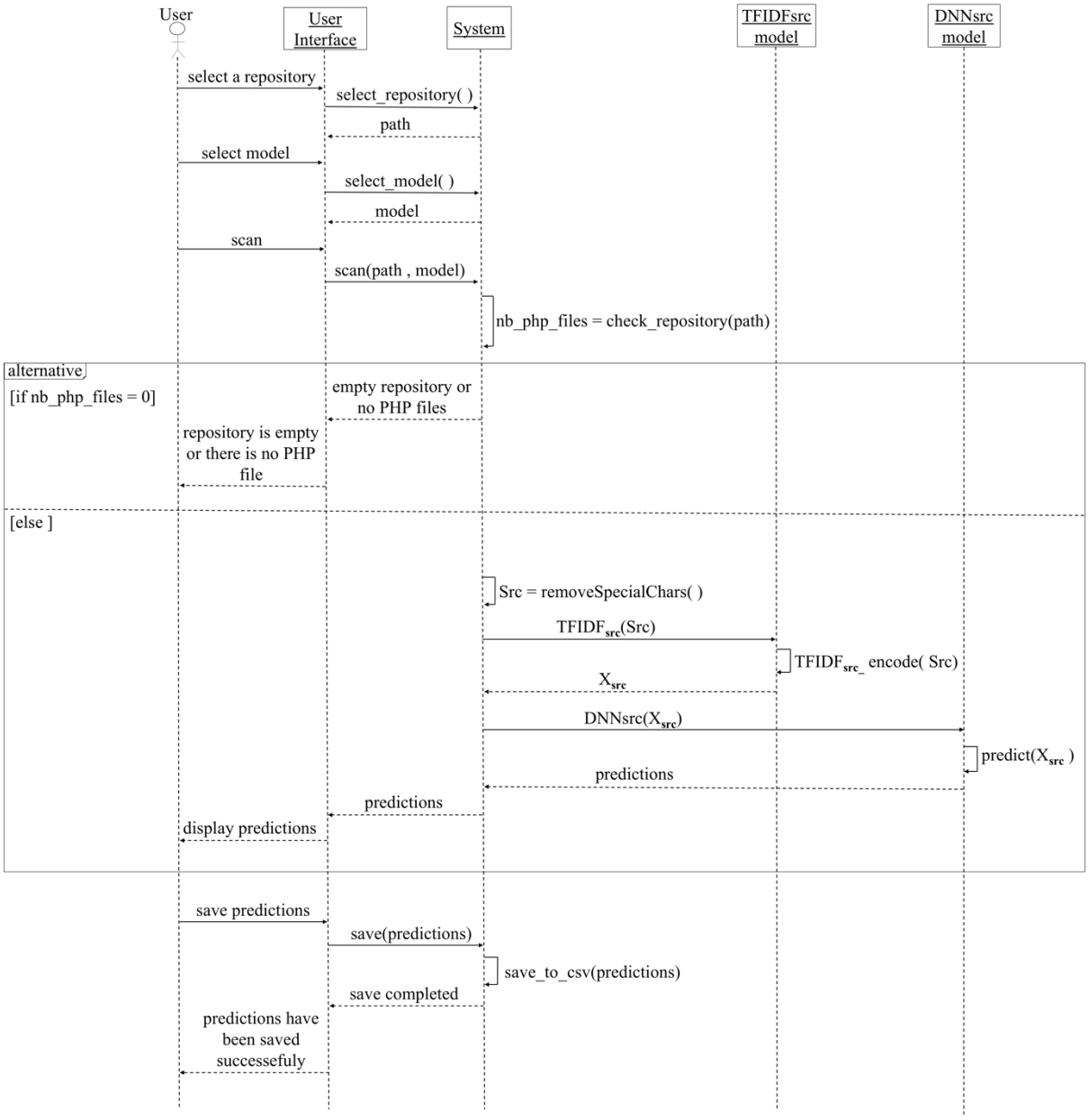

Figure 5.3. Diagramme de séquence : cas d'un modèle d'apprentissage profond DNN_{src}.

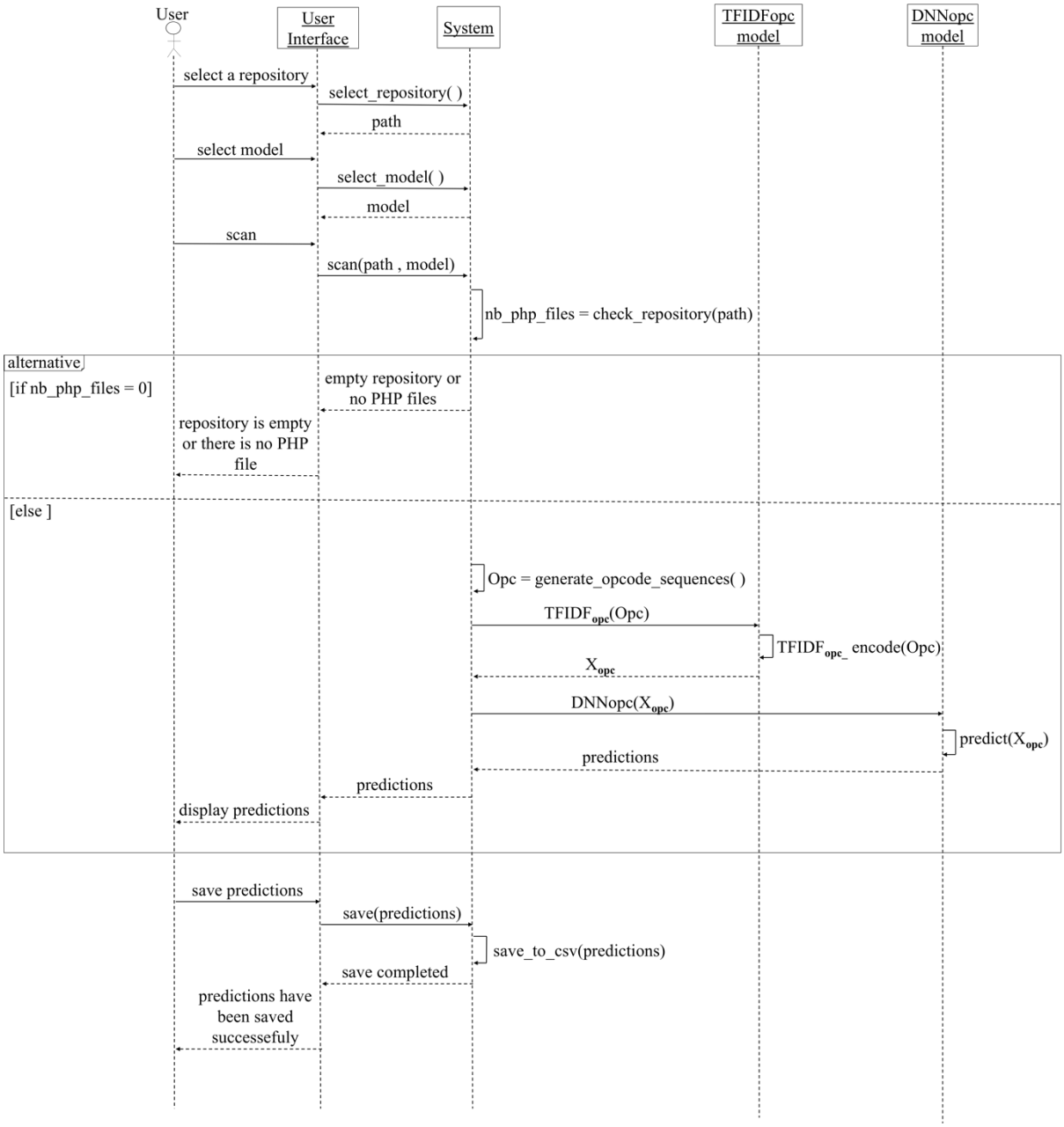

Figure 5.4. Diagramme de séquence : cas d'un modèle d'apprentissage profond DNN_{opc}.

5.1.2.3. Cas des modèles hybrides (combinés) :

Lorsque l'utilisateur choisit l'un des modèles hybrides ($RF\text{-}DNN_{src}$ ou $RF\text{-}DNN_{opc}$), un processus compliqué sera appliqué. Par soucis de simplicité, nous montrons les

diagrammes de séquences des cas RF-DNN_{src} RF-DNN_{opc} dans les figures 5.5 et 5.6 respectivement.

Après la vérification du répertoire choisi par l'utilisateur et dans les deux cas, le système va générer le vecteur de caractéristiques de chaque fichier et va envoyer ces vecteurs au modèle RF. Après la récupération de la prédiction du modèle RF, le système procède selon le cas :

- **Cas de RF-DNN_{src} :** nettoie le contenu de chaque fichier PHP, appelle le modèle $TFIDF_{src}$ pour la construction des vecteurs de caractéristiques, qui seront envoyés au modèle DNN_{src} pour un deuxième tour d'analyse.
- **Cas de RF-DNN_{opc} :** génère les séquences d'opcodes de chaque fichier et les envoie au modèle $TFIDF_{opc}$ pour la construction des vecteurs de caractéristiques. Ces vecteurs sont envoyés au modèle DNN_{opc} pour un deuxième tour d'analyse.

Par la suite, le système construit une matrice depuis les prédictions des deux modèles RF et DNN_{src} ou DNN_{src}; cette nouvelle matrice est envoyée à un autre modèle RF (noté dans les figures par RF-Stack model), pour un tour final d'analyse. Les résultats de ce nouveau modèle RF seront adoptés et affichés à l'utilisateur pour être consulté et sauvegardé en cas de besoin.

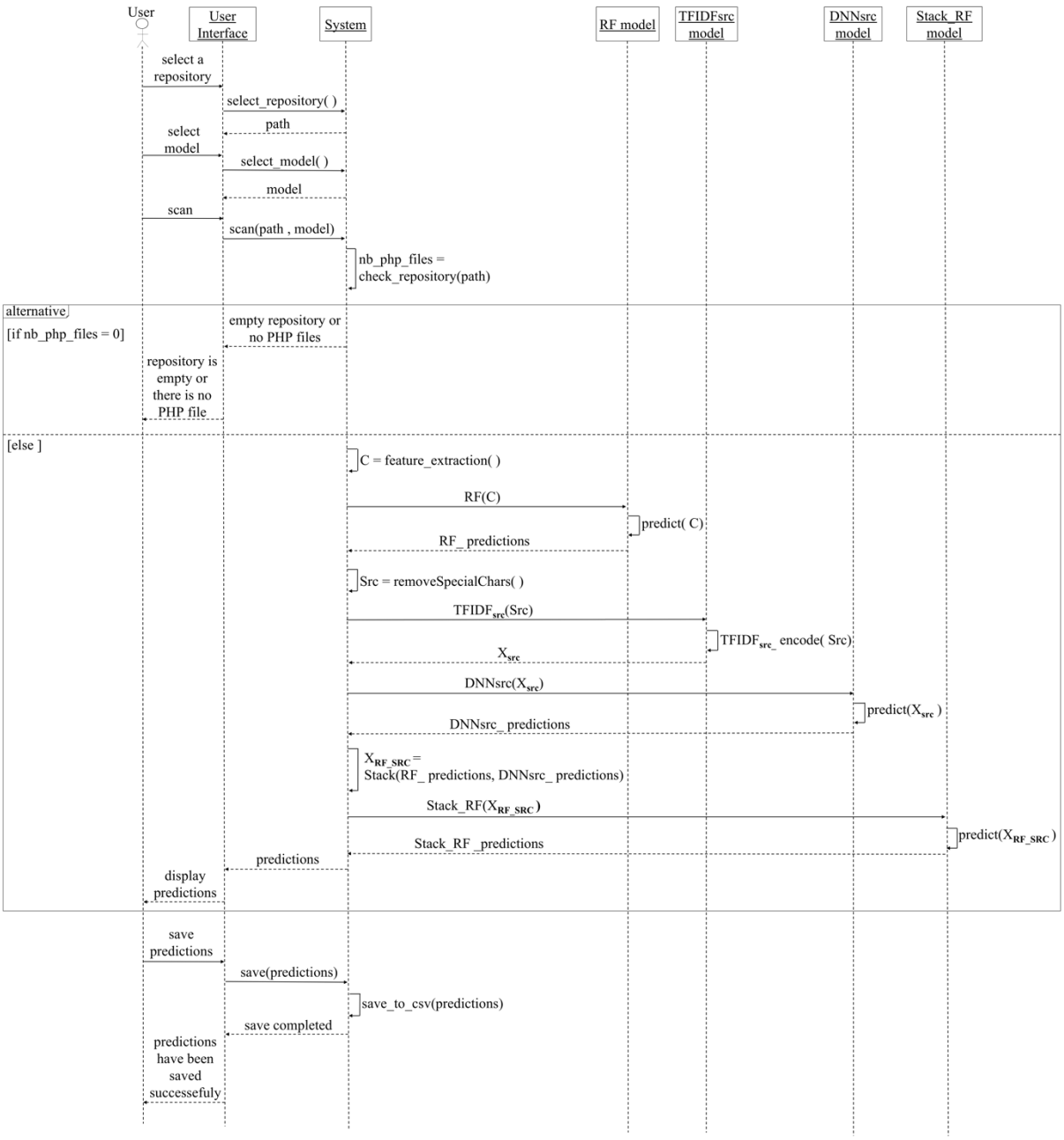

Figure 5.5. Diagramme de séquence : cas d'un modèle hybride RF-DNN$_{src}$.

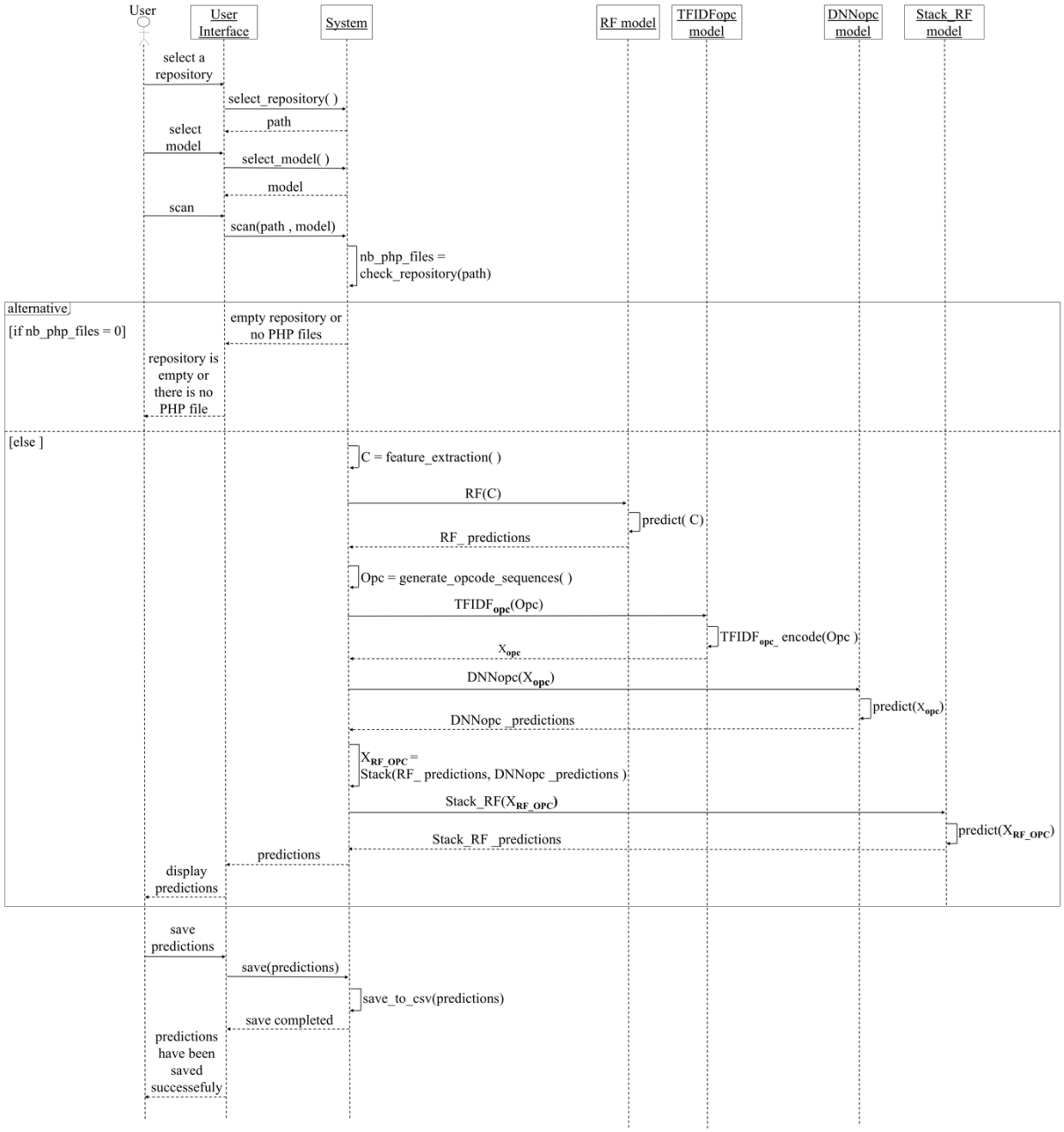

Figure 5.6. Diagramme de séquence : cas d'un modèle hybride RF-DNN_{opc}.

La figure 5.7 montre le diagramme de séquence montrons le cas de DNN^2. Pour une simplification, nous avons réduit le nombre des acteurs dans le diagramme. L'acteur DNN joue le rôle des deux modèles DNN_{src} et DNN_{opc} alors que TFIDF remplace les deux modèles de vectorisation $TFIDF_{src}$ et $TFIDF_{opc}$. Dans ce dernier cas, le système procède par le nettoyage des sources des fichiers, la vectorisation en appelant $TFIDF_{src}$ et un premier tour de prédiction en appelant DNN_{src}. Puis, le système lance un

deuxième tour de prédiction en appelant DNN_{opc} après la génération des séquences d'opcodes et vectorisation de ces séquences en appelant $TFIDF_{opc}$. Les prédictions des deux modèles sont combinées et envoyées à un modèle ensembliste RF (noté par RF-Stack). Les prédictions de modèles RF sont finales et seront affichées à l'utilisateur.

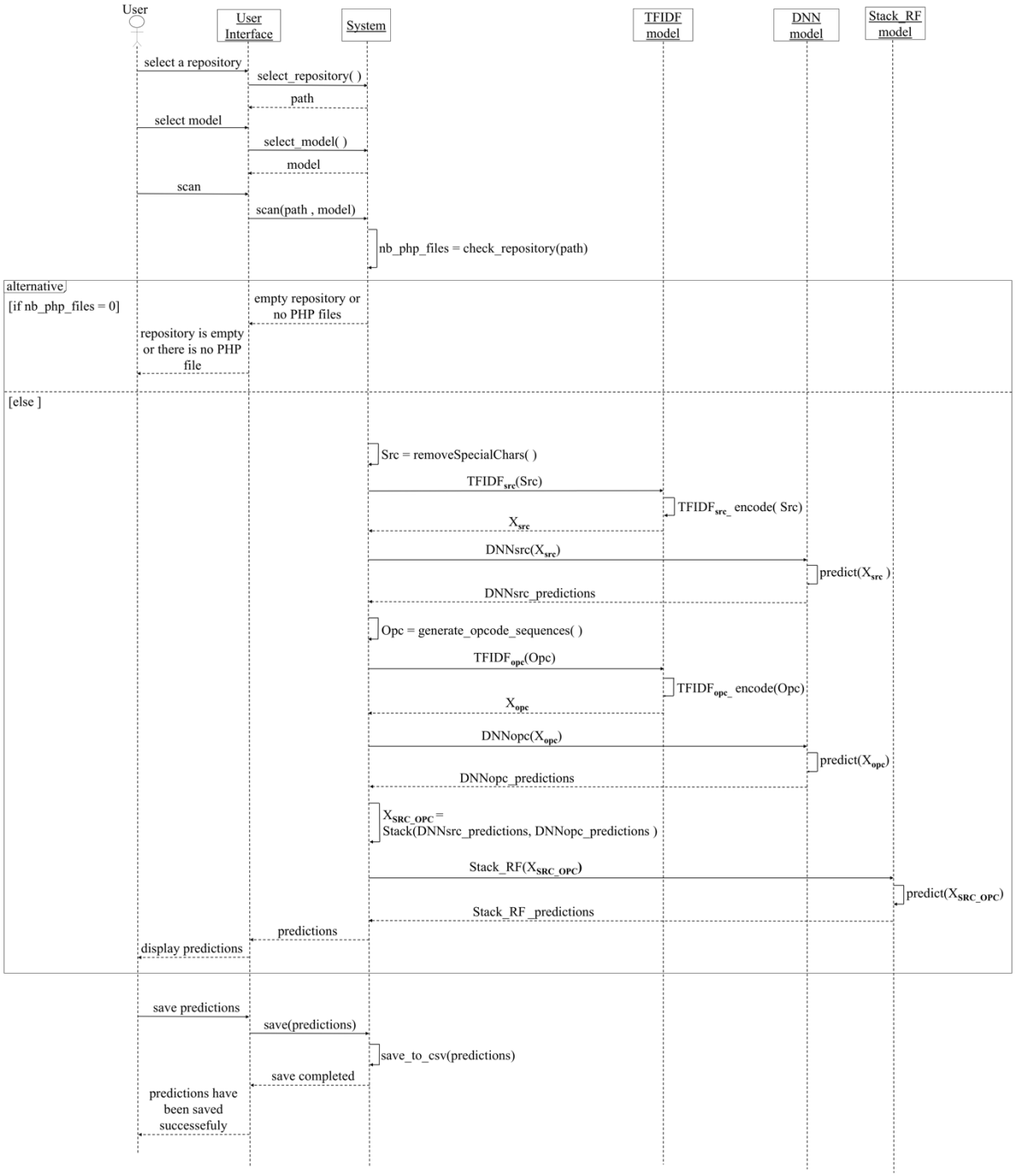

Figure 5.7. Diagramme de séquence : cas d'un modèle hybride DNN^2.

5.2. Implémentation de l'outil

Pour la réalisation de notre projet, et spécifiquement la conduite des expérimentations et le développement des différents modèles et l'outil graphique, nous avons utilisé différents outils d'aides, langages et plateformes de développement. Dans cette section, nous présentons chacun de ces outils et plateformes.

5.2.1. Plateforme PyCharm

Pycharm est un éditeur de code très pratique qui permet de créer des projets et d'éditer des scripts python. Cet éditeur est développé par l'entreprise JetBrains. Ce logiciel est multiplateforme, c'est-à-dire qu'il fonctionne sous les différents systèmes d'exploitation, en plus d'être compatible avec plusieurs versions de python. La plateforme PyCharm est disponible en deux éditions, une édition professionnelle diffusée sous licence propriétaire et une édition communautaire diffusée sous licence Apache. Dans notre projet, nous avons utilisé l'édition communautaire version 2021.1.

5.2.2. Langage Python

Python est un langage de programmation libre de haut niveau, créé par Guido van Rossum, sa première publication a été en 1991. Python est un langage interprété, c'est-à-dire qu'il peut être exécuté sans compilation. Il est caractérisé par un système de typage dynamique, une gestion de mémoire automatique et une bibliothèque multifonctionnelle complète. Ce langage prend en charge plusieurs paradigmes de programmation, et il est adaptable pour tous les systèmes d'exploitation. Python est un langage très approprié pour les apprenants débutants, mais il est aussi très motivant pour les utilisateurs expérimentés [32]. Pendant la réalisation de notre projet, nous avons utilisé la version 3.6 ainsi qu'un ensemble de bibliothèques utiles. Nous classons ces bibliothèques selon leurs rôles dans la réalisation de notre projet en 4 catégories : *gestion de l'ensemble de données*, *extraction des caractéristiques*, *gestion des modèles d'apprentissage* et *développement de l'interface graphique*.

5.2.2.1. Gestion de l'ensemble de données :

pandas (v.1.1.5) : Pandas est une bibliothèque python qui fournit des structures de données complexes et des fonctions d'analyse de données. Elle se base sur l'utilisation d'une structure nommée *dataframe*, qui est une structure de données tabulaire bidimensionnelle [33]. Nous avons utilisé cette bibliothèque pour stocker notre ensemble de données en mémoire.

numpy (v.1.19.5) : Numpy est la bibliothèque responsable du calcul mathématiques en python, elle se base sur l'utilisation des tableaux de N dimensions. Ces tableaux sont très rapides, et facilitent les opérations mathématiques sur un grand nombre de données [33]. Nous avons utilisé cette bibliothèque pour appliquer des opérations mathématiques dans la phase de traitement de l'ensemble de données.

5.2.2.2. Extraction des caractéristiques :

neopi : est un module Python, qui se base sur l'utilisation de plusieurs méthodes statistiques dans le but de détecter le contenu obscurci et crypté des fichiers texte et des scripts. Ce script fonctionne sur tous les systèmes sur lequel est installé python 2.6 ou plus [W7]. Dans notre projet, Neopi nous a aidé pour extraire les sept caractéristiques statistiques C1-7 (Voir chapitre IV), que nous avons utilisé pour expérimenter les modèles d'apprentissage traditionnels comme RF.

regex (v.13.1.2020) : une bibliothèque d'expressions régulières très utile dans l'analyse des textes. Elle offre des algorithmes de recherche de chaînes de caractères et de séquences correspondantes à des patrons spécifiés suivant des règles plus précises. Dans notre travail, nous avons utilisé cette bibliothèque pour le nettoyage des scripts (suppression des caractères spéciaux) et l'extraction des séquence d'opcodes (modts en majuscules depuis des textes non structurés).

subprocess : est un module de python qui permet de créer de nouveaux processus et de se connecter à leurs canaux d'entrée, de sortie ou d'erreur. Nous avons utilisé ce

module pour appeler des commandes systèmes, spécifiquement l'interpréteur de langage PHP pour la génération des séquences d'opcodes.

sklearn (v.24.2): est une bibliothèque qui comporte plusieurs sous modules. Dans la phase d'extraction des caractéristiques, nous avons utilisé le sous module *sklearn.feature_extraction*. Il est utilisé pour construire les modèles TFIDF en appelant l'algorithme TfidfVectorizer. Ce dernier a été utilisé pour convertir des textes à des vecteurs de caractéristiques en appliquant la méthode TFIDF (Voir Chapitre IV).

5.2.2.3. Gestion des modèles d'apprentissage :

sklearn (v.24.2): Cette fois-ci, nous avons utilisé la bibliothèque sklearn pour importer des sous modules, qui fourniront les différents algorithmes d'apprentissage automatique. Nous avons utilisé la liste des algorithmes suivants :

- RandomForestClassifier()
- tree.DecisionTreeClassifier()
- SVC()
- neighbors.KNeighborsClassifier()
- SGDClassifier()
- GaussianNB()
- BernoulliNB()
- MLPClassifier()
- LogisticRegression()

keras (v.2.4.3) : est une bibliothèque de haut niveau qui fournit les concepts fondamentaux de l'apprentissage profond telle que : la création des couches pour les réseaux de neurones. Cette bibliothèque s'exécute au-dessus de *TensorFlow*. L'une des méthodes d'utilisation la plus courantes dans keras est l'API séquentielle, qui se base sur la création d'une séquence de couches. Ce modèle séquentiel est considéré comme une pile linéaire de couches [34]. Nous avons utilisé keras pour la construction des deux modèles d'apprentissage profond DNN_{src} et DNN_{opc}.

Tensorflow (v.2.4.1) : est une bibliothèque open source créé par Google spécialement pour l'apprentissage profond. Cette bibliothèque fournit l'utilisation et l'intégration des modèles dans des programmes ou des applications. Cette bibliothèque est nécessaire pour l'exécution des modèles construits par keras.

5.2.2.4. Développement de l'interface graphique :

PySimpleGUI (v.4.45.0) : est une bibliothèque qui permet de créer des interfaces graphiques en python de haute qualité. Cette bibliothèque a été créée en 2018, elle englobe les mêmes apparences et fonctionnalités que la bibliothèque de *Tkinter*. La bibliothèque PySimpleGUI a plusieurs propriétés tels que :

- Permettre aux développeurs d'obtenir le résultat requis en utilisant le moins de code possible, et de créer à la fois des interfaces des applications de bureau et des applications web.
- Capable d'interagir avec les différentes bibliothèques de python.
- Fonctionne sous la version python 3 et plus.

joblib (v.1.0.1) : fournit un ensemble d'outils qui permettent de réaliser des tâches légères en pipeline en Python. Dans notre application, nous avons utilisé joblib pour sauvegarder et charger nos modèles dans le but de les réutiliser dans notre application.

5.3. Mode d'utilisation de « PHP Webshell Scanner »

L'outil PHP Webshell scanner est constitué d'une seule fenêtre principale montrée dans la figure 5.8. À partir de cette fenêtre, l'utilisateur peut :

1. Sélectionner un répertoire à analyser en cliquant sur le bouton *Browse* et le chemin du répertoire s'affiche dans le champ d'édition qui lui correspond.
2. Choisir un modèle d'apprentissage automatique depuis une liste déroulante.
3. Lancer le processus d'analyse en cliquant sur le bouton *Scan*.
4. Visualiser les étapes intermédiaires de l'analyse sur le champ *Console*.

5. Consulter les résultats finales de l'analyse qui s'affichent sur le tableau *Output*. Pour faciliter la visualisation des résultats, nous adoptons deux couleurs : rouge pour les fichiers Webshells et vert pour les fichiers normaux (Voir figure 5.9).
6. Sauvegarder les résultats de la prédiction dans des fichiers CVS afin de les utiliser pour une comparaison avec d'autres modèles.
7. Vider explicitement les blocs *Console* et *Output* en cliquant sur les boutons *Clear console* et *Clear output* respectivement.
8. Lire une brève description de l'outil et ses développeurs en cliquant sur le bouton *About*.
9. Quitter l'application en cliquant sur le bouton *Exit*.

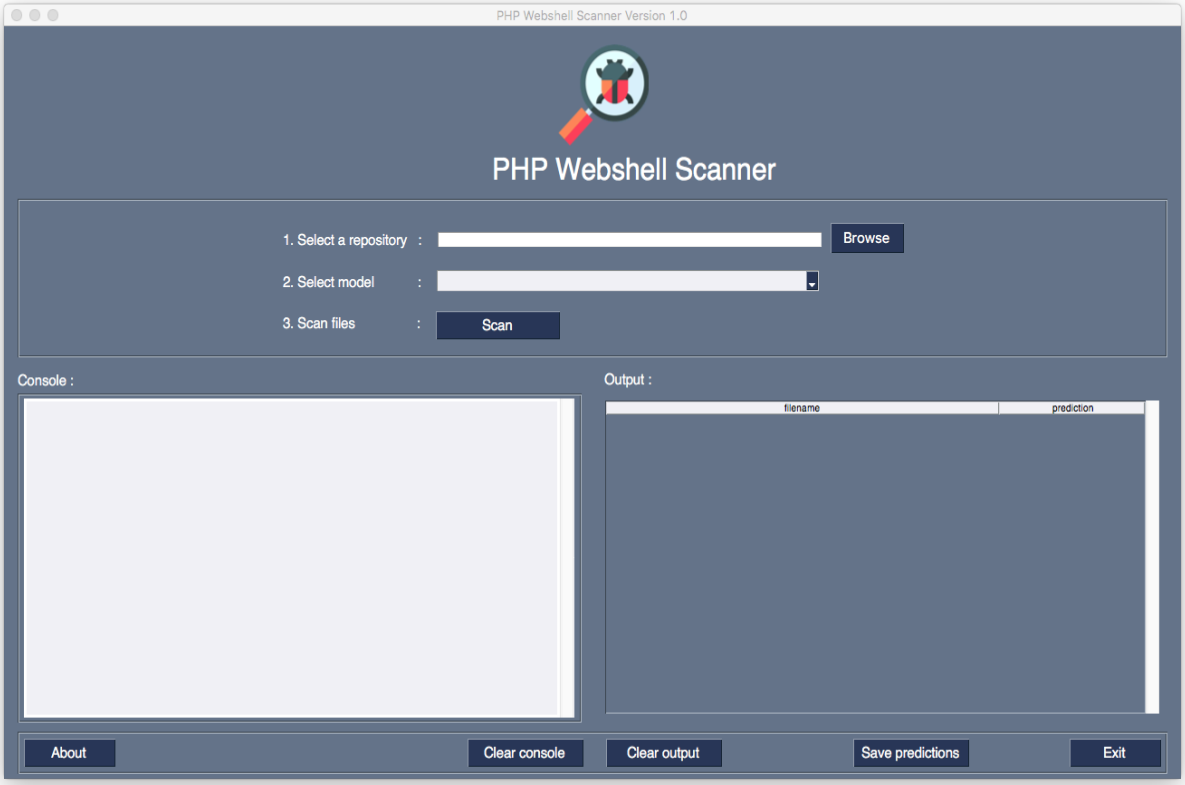

Figure 5.8. Fenêtre principale de PHP Webshell Scanner

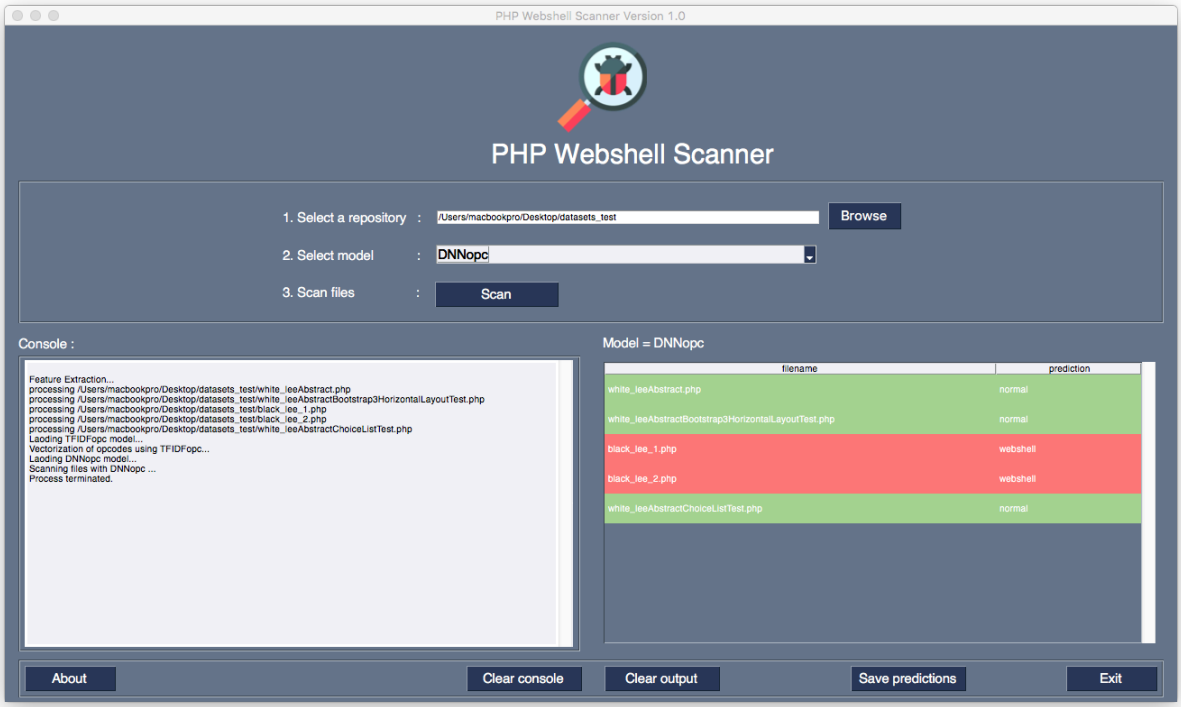

Figure 5.9. Affichage des résultats par PHP Webshell Scanner

5.4. Conclusion

Dans ce chapitre, nous avons présenté les différents outils de développements utilisés pendant la réalisation de notre projet où nous avons utilisé la méthodologie UML dans la phase de conception. Nous avons aussi présenté l'interface graphique de notre application en montrant son mode d'utilisation.

CONCLUSION GENERALE

CONCLUSION GENERALE

Avec le développement de la technologie web et l'augmentation explosive de la quantité de l'information, la sécurité web devient de plus en plus importante. Les vulnérabilités du web font partie des problèmes de sécurité ; les attaquants exploitent ces vulnérabilités pour télécharger des codes malveillants sur les serveurs web. Ce code nocif est appelé webshell. Une fois l'interaction réalisée avec le webshell, ce dernier permet à l'attaquant d'avoir tous les droits administratifs sur le serveur ciblé, par un contrôle à distance. Par conséquent, il est très important de détecter avec précision et efficacité les webshells malveillants dans les serveurs web.

Dans ce cadre, nous avons proposé une approche pour la détection des webshells malveillants écrits en PHP. Cette approche se base sur l'utilisation des modèles d'apprentissage profond. Nous avons aussi examiné d'autres modèles de détection des webshells qui se basent sur l'apprentissage traditionnel. La réalisation de ces modèles a été faite à partir d'un ensemble de données solide et très propre. Après plusieurs expérimentations de ces modèles, nous avons constaté que les modèles d'apprentissage profond, notamment celui proposé dans ce projet, sont plus performants que les modèles d'apprentissage automatique (les algorithmes traditionnels). De là, nous pouvons conclure que l'utilisation de l'apprentissage profond est plus efficace par rapport à l'utilisation des algorithmes traditionnels de l'apprentissage automatique dans la détection des webshells écrits en PHP.

Au terme de ce travail, nous espérons continuer dans ce domaine en généralisant notre approche pour inclure d'autres langages de scripts, telles que : ASP, ASPX, JSP et PERL-CGI. Nous espérons aussi avoir la possibilité d'examiner la nouvelle technique d'apprentissage profond qui s'appelle « *Transformers* » ou « transformateurs » dans le domaine de détection des webshells.

BIBLIOGRAPHIE & WEBOGRAPHIE

BIBLIOGRAPHIE

[1] Xin Sun, Xindai Lu, and Hua Dai, *A Matrix Decomposition based Webshell Detection Method*, In Proceedings of the 2017 International Conference on Cryptography, Security and Privacy (ICCSP '17). ACM, New York, NY, USA, pp. 66–70, 2017.

[2] Wu, Y., Sun, Y., Huang, C., Jia, P., Liu, L. *Session-Based Webshell Detection Using Machine Learning in Web Logs*. Security and Communication Networks. Vol. 2019, pp. 1-11, 2019.

[3] Zhu T, Weng Z, Fu L, Ruan L. *A Web Shell Detection Method Based on Multiview Feature Fusion*. Journal of Applied Sciences, 10(18), pp. 1-16, 2020.

[4] Guo Y, Marco-Gisbert H, Keir P, *Mitigating Webshell Attacks through Machine Learning Techniques*. Journal of Future Internet, 12(1), pp. 1-16, 2020.

[5] Qi L., Kong R., Lu Y. and Zhuang H., *An end-to-end detection method for Webshell with deep learning*, in 2018 Eighth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC), IEEE CS, pp. 660–665, 2018.

[6] Amazon Web Services, *Présentation des processus de sécurité*, Technical Report, AWS, Aaût 2015.

[7] Shai Shalev-Shwartz and Shai Ben-David, *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, USA, 2014.

[8] Moualek Djaloul Youcef, *Deep Learning pour la classification des images,* Mémoire Master, Département d'informatique, Université de Tlemcen, Algérie, 2017.

[9] Nikhil Buduma and Nicholas Locascio, *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms (1st. ed.),* O'Reilly Media, Inc., 2017

[10] Ravì Daniele, Wong Charence, Deligianni Fani, Berthelot Melissa, Andreu-Perez Javier, Lo Benny and Yang Guang-Zhong, *Deep Learning for Health Informatics*, in IEEE Journal of Biomedical and Health Informatics, 21(1), pp. 4-21, 2017

[11] Slimani Massinissa et Khaled Adel, *Reconnaissance Automatique des chiffres avec le Deep Learning*, Mémoire Master, Département de génie électrique, Université de Bejaïa, Algérie, 2020

[12] Soumia Madoui, L'utilisation du Deep Learning pour l'extraction du contenu des pages web, Mémoire Master, Département Informatique, Université de Biskra, Algérie, 2019

[13] Héritier Nsenge Mpia and Inipaivudu Baelani Nephtali, *Gradient Back-Propagation Algorithm in the Multilayer Perceptron: Foundations and Case Study*, International Journal of Innovation and Applied Studies, 32(2), pp. 271-290, 2021.

[14] Farhadi Farnoush, *Learning activation functions in deep neural networks,* Mémoire de Maîtrise en sciences appliquées, Département de Mathématiques et de Génie Industriel, École Polytechnique de Montréal, Montréal, Cannada, 2017

[15] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò and Roberto Prevete. *A survey on modern trainable activation functions.* Neural Networks, vol.138, pp. 14-32, 2021

[16] Jojo Moolayil. *Learn Keras for Deep Neural Networks: A Fast-Track Approach to Modern Deep Learning with Python (1st. ed).* Apress, USA, 2018

[17] Navin Kumar Manaswi. *Deep Learning with Applications Using Python: Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras (1st. ed).* Apress, USA, 2018

[18] Bo-Sung Kwon, Rae-Jun Park and Kyung-Bin Song. *Short-Term Load Forecasting Based on Deep Neural Networks Using LSTM Layer*. Journal of Electrical Engineering & Technolology. Vol. 15**,** pp. 1501–1509, 2020

[19] Althelaya Khaled A. and El-Alfy El-Sayed M. and Mohammed Salahadin, *Stock Market Forecast Using Multivariate Analysis with Bidirectional and Stacked (LSTM, GRU),* In 21st Saudi Computer Society National Computer Conference (NCC), Riyadh, Saudi Arabia, IEEE CS, pp. 1-7, 2018

[20] Thippa Reddy G., Swarna Priya R.M., Parimala M., Chiranji Lal Chowdhary, Praveen Kumar Reddy M., Saqib Hakak and Wazir Zada Khan, *A deep neural networks based model for uninterrupted marine environment monitoring*, Computer Communications, Vol. 157, pp. 64-75, 2020

[21] Michael W Browne, *Cross-Validation Methods*, Journal of Mathematical Psychology, 44(1), pp. 108-132, 2000.

[22] Abdelhakim Hannousse and Salima Yahiouche, *Handling webshell attacks: A systematic mapping and survey*, Computers & Security, Vol. 108, pp. 1-26, 2021.

[23] Weiqing Huang, Chenggang Jia, Min Yu, Kam-Pui Chow, Jiuming Chen, Chao Liu and Jianguo Jiang, *Enhancing the Feature Profiles of Web Shells by Analyzing the Performance of Multiple Detectors*. In Peterson G., Shenoi S. (eds) Advances in Digital Forensics XVI. DigitalForensics. IFIP Advances in Information and Communication Technology, Vol 589. Springer, Cham, pp. 57-72, 2020.

[24] Yong Fang, Yaoyao Qiu, Liang Liu, and Cheng Huang, *Detecting Webshell Based on Random Forest with FastText*. In Proceedings of the 2018 International Conference on

Computing and Artificial Intelligence (ICCAI 2018). Association for Computing Machinery, New York, NY, USA, pp. 52–56, 2018.

[25] Tao Fangjian, Cao Chunjie and Liu Zhihui, *Webshell Detection Model Based on Deep Learning*. In Sun X., Pan Z., Bertino E. (eds) Artificial Intelligence and Security. ICAIS 2019. Lecture Notes in Computer Science, Vol. 11635. Springer, Cham, pp. 408-420, 2019.

[26] Zhuo-Hang Lv, Han-Bing Yan and Mei Rui, *Automatic and Accurate Detection of Webshell Based on Convolutional Neural Network*. In Yun X. et al. (eds) Cyber Security. CNCERT 2018. Communications in Computer and Information Science, Vol. 970. Springer, Singapore, pp. 73-85, 2019.

[27] Ai Zhuang, Nurbol Luktarhan, Ai Jun Zhou, and Dan Lv, *WebShell Attack Detection Based on a Deep Super Learner*. Symmetry 12(9), pp. 1-16, 2020.

[28] Binbin Yong, Wei Wei, Kuan-Ching Li, Jun Shen, Qingguo Zhou, Marcin Wozniak, Dawid Połap and Robertas Damaševičius, *Ensemble machine learning approaches for webshell detection in Internet of things environments*. Transactions on Emerging Telecommunications Technologies, pp. 1-12, 2020.

[29] Djaghout Yahia et Grini Achraf, *Développement d'un benchmark pour les détecteurs des webshells malveillants.* Mémoire de find d'étude License, département d'informatique, Université 8 Mai 1945, Guelma, Juillet 2021.

[30] Soleas Agisilaos, *Detecting malicious code in a web server***,** PhD Thesis, Department of digital systems, University of PIRAEUS, Athens, Greece, 2016.

[31] Alan Dennis, Barbara Haley Wixom, and David Tegarden. *Systems Analysis and Design with UML*. 4th Edition, Wiley Publishing, 2012.

[32] Younes Derfoufi. *Programmation en langage Python.* Support de cours de CRMEF OUJDA, Maroc, 2019.

[33] Fabio Nelli. *Python Data Analytics: With Pandas, NumPy, and Matplotlib.* 2nd. Edition, Apress, USA, 2018.

[34] Manaswi Navin Kumar Manaswi. *Understanding and Working with Keras*. Chapter 1 of Deep Learning with Applications Using Python. Apress, Berkeley, CA, pp; 31-43, 2018.

WEBOGRAPHIE

[W1] Microsoft 365 Defender Research Team, *Web shell attacks continue to rise*, February 2021, https://www.microsoft.com/security/blog/2021/02/11/web-shell-attacks-continue-to-rise/, [consulté Juin 2021]

[W2] Web Technology Surveys, *Usage statistics of server-side programming languages for websites*, June 2021, https://w3techs.com/technologies/overview/programming_language, [consulté Juin 2021].

[W3] Luke Leal, *Webshell in Fake Plugin /blnmrpb/ Directory*, January 2020, https://blog.sucuri.net/2020/01/webshell-in-fake-plugin-blnmrpb-directory.html, Juanuary 2020, [consulté Juin 2021].

[W4] Universalis encyclopedie *Apprentissage profond ou deep learning, Réseaux de neurones formels*, https://www.universalis.fr/encyclopedie/apprentissage-profond-deep-learning/2-reseaux-de-neurones-formels/, [consulté Mai 2021].

[W5] Ebergementwebs, *Qu'est-ce que l'apprentissage automatique et pourquoi Important ?,* https://www.hebergementwebs.com/nouvelles/qu-est-ce-que-l-apprentissage-automatique-et-pourquoi-est-il-important, [consulté Mai 2021].

[W6] Lucas Scott, *Data Preparation for Machine Learning: The Ultimate Resource Guide*, 2020, https://lionbridge.ai/articles/data-preparation-for-machine-learning-the-ultimate-resource-guide/, [consulté Mai 2021].

[W7] Scott Behrens and Ben Hagen, *Web Shell Detection Using NeoPI*, Digital forensics, 2011, https://resources.infosecinstitute.com/topic/web-shell-detection/, [consulté Août 2021].