الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieure et de la recherche scientifique

Centre Universitaire Abdelhafid Boussouf – Mila
Institut des Mathématiques et informatique
Département de l'Informatique

Spécialité : Sciences et technologies de l'information et de la communication (STIC)

La sécurité des applications web basée sur l'apprentissage automatique

Réalisé par :

HAMIDA Amira **SIKHA Chatila**

Soutenue devant le jury :

- **Président : M. MERABET Adil M.A.A Centre universitaire de Mila**
- **Examinateur : M. ATTIA Mourad M.A.A Centre universitaire de Mila**
- **Encadrant : M. SELMANE Samir M.A.A Centre universitaire de Mila**

Année universitaire : 2024/2025

Dédicaces

Nous dédions ce travail à notre établissement, le Centre Universitaire de Mila, et espérons qu'il constituera une valeur ajoutée à la bibliothèque de notre département et qu'il contribuera à l'enrichissement de futurs travaux.

Amira Djihane

Dédicaces

Je tiens à exprimer ma profonde gratitude à mes chers parents, à qui je dois l'essentiel de ma réussite. Leur amour, leurs sacrifices et leur soutien constant ont été le socle solide de mon parcours.

Je remercie également mes frères et sœurs pour leur présence, leur aide et leurs encouragements tout au long de cette aventure, avec une pensée toute particulière à Abdellah.

Ma reconnaissance va aussi à mon encadrant M. Selmane Samir, pour sa précieuse orientation, sa patience et son accompagnement constant.

Je n'oublie pas non plus de remercier chaleureusement mes camarades de classe, qui ont su rendre ce parcours plus léger et motivant grâce à leur présence bienveillante, leur entraide et l'ambiance agréable qu'ils ont créée au quotidien.

Enfin, je remercie l'ensemble des enseignants et toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce travail.

Chatila

Remerciements

Nous tenons à remercier l'ensemble de responsables de département des sciences et technologies du Centre Universitaire Mila pour la formation dispensée.

Nos vifs remerciements s'adressent à notre encadrant M. SELMANE Samir pour ses précieux conseils et orientations.

Aussi, nous remercions les membres du jury M MERABET Adil et M. ATTIA Mourad d'avoir consacré de leur temps à la lecture, l'évaluation et l'ajustement du contenu de ce travail.

N'oublions pas de remercier toute personne qui a contribué de près ou de loin dans la réalisation du présent mémoire.

Résumé

Avec la croissance rapide des applications web, les plateformes en ligne sont devenues des cibles privilégiées pour de nombreuses attaques. Les méthodes traditionnelles de sécurité, bien qu'encore utilisées, montrent aujourd'hui leurs limites face à des menaces de plus en plus sophistiquées. Ce mémoire explore l'intégration de l'apprentissage automatique comme solution intelligente pour renforcer la sécurité des applications web. Dans une première partie, nous avons présenté les principales vulnérabilités web (injection SQL, XSS, CSRF, DoS, etc.) ainsi que les techniques classiques de protection. Ensuite, nous avons étudié les apports de l'apprentissage automatique à travers une sélection de datasets spécialisés et des méthodes de prétraitement rigoureuses. Dans la partie expérimentale, plusieurs modèles de classification ont été appliqués (Random Forest, LightGBM, XGBoost) sur deux datasets (CSIC_HTTPParams et SR-BH 2020), permettant d'évaluer leur efficacité dans la détection des attaques. Enfin, une expérimentation réelle a été réalisée dans un environnement local simulé, permettant de tester le système dans des conditions proches du contexte opérationnel. Les résultats obtenus confirment que l'apprentissage automatique constitue une approche prometteuse pour améliorer les mécanismes de détection d'intrusion dans les applications web.

Mots-clés : Sécurité des applications web, Attaques web, Apprentissage automatique, Prétraitement des données, Systèmes intelligents de détection,

Abstract

With the rapid expansion of web applications, online platforms have become prime targets for various cyberattacks. Traditional security mechanisms, while still in use, have shown their limitations in the face of increasingly sophisticated threats. This work aims to explore the use of machine learning as an intelligent approach to strengthening the security of web applications. We began by reviewing the main security vulnerabilities affecting web applications, such as SQL Injection, XSS, CSRF, and Denial-of-Service (DoS) attacks, as well as classical protection techniques. Then, we focused on the application of machine learning by selecting relevant datasets and implementing rigorous data preprocessing steps. In the experimental part, several classification models (such as Random Forest, LightGBM, and XGBoost) were applied to two datasets (CSIC_HTTPParams and SR-BH 2020) to evaluate their effectiveness in detecting attacks. Additionally, a real-world simulation was conducted in a local environment to test the proposed system under near-operational conditions. The results confirm that machine learning is a promising solution for improving intrusion detection mechanisms and enhancing the overall security of web applications.

Keywords: Web application security, Cyberattacks, Machine learning, Data preprocessing, Intelligent detection systems

الملخص

مع التوسع السريع في استخدام تطبيقات الويب، أصبحت المنصات الإلكترونية هدفًا رئيسيًا للعديد من الهجمات السيبرانية. وقد أظهرت الأساليب الأمنية التقليدية محدوديتها في مواجهة التهديدات المتزايدة والمعقدة، مما استدعى البحث عن حلول أكثر ذكاءً وتطورًا. يهدف هذا العمل إلى دراسة إمكانية تعزيز أمن تطبيقات الويب بالاعتماد على تقنيات التعلم الآلي .في البداية، قمنا باستعراض أبرز الثغرات الأمنية التي تهدد تطبيقات الويب، مثل هجمات الحقن SQL، وXSS، وCSRF، DoS، إضافة إلى عرض آليات الحماية الكلاسيكية. ثم انتقلنا إلى توظيف تقنيات التعلم الآلي، من خلال اختيار قواعد بيانات متخصصة وتطبيق خطوات معالجة مسبقة دقيقة. وفي الجانب التجريبي، قمنا بتطبيق مجموعة من نماذج التصنيف مثل Random Forest و LightGBMو XGBoostعلى مجموعتي بيانات CSIC_HTTPParamsو SR-BH 2020بهدف تقييم فعاليتها في كشف الهجمات .كما تم تنفيذ تجربة واقعية في بيئة محلية تحاكي ظروف الاستخدام الفعلي، من أجل اختبار النظام في سياق أقرب إلى الواقع العملي. وقد أظهرت النتائج أن تقنيات التعلم الآلي تُعد أداة واعدة لتعزيز آليات كشف التسلل وتأمين تطبيقات الويب بشكل أكثر فاعلية.

الكلمات المفتاحية: أمن تطبيقات الويب، هجمات الويب، التعلم الآلي، المعالجة المسبقة للبيانات، الكشف الذكي عن الهجمات.

Table des matières

Résumé VI
Abstract VII
الملخص VIII
Table des matières IX
Liste des figures XII
Liste des tableaux XIV
Abréviations XV
Introduction générale 2
Chapitre 1 : Généralités sur la sécurité des applications web 4
Introduction 5
1. Attaque par injection SQL(SQL injection) 5
1.1 Principe 5
1.2 Exemple d'attaque SQL injection 6
1.3. Méthodes traditionnelles de protection 6
2. Attaque par Cross Site Request Forgery (CSRF) 7
2.1. Principe 7
2.2. Exemple d'attaque CSRF 8
2.3. Méthodes traditionnelles de protection 8
3. Attaques par Déni de Service (DOS/DDOS) 9
3.1. Principe 9
3.2. Méthodes traditionnelles de protection 10
4. Attaque par Cross-Site Scripting (XSS) 11
4.1. Principe 11
4.2. Exemples d'attaques XSS 13
4.2.1. Attaque XSS persistant 13
4.2.2. Attaque XSS non persistant 13
4.3. Méthode traditionnelle de protection 14
5. Attaque par Man-in-the-Middle (MitM) 15
5.1. Principe 15
5.2. Exemple d'attaque MitM 16
5.3. Méthodes traditionnelles de protection 17
6. Attaque par Brute Force et Credential Stuffing 17
6.1. Principe 17
6.2. Exemples d'attaques Brute Force et Credential Stuffing 18
6.2.1. Attaque Credential Stuffing 18
6.2.2. Attaque Brute Force 19
6.3. Méthodes traditionnelles de protection 19
7.Attaque par XML External Entity (XEE) 21
7.1. Principe 21
7.2. Exemple d'attaque XEE 22
7.3. Méthode traditionnelle de protection 23
8. Attaque par Insecure Direct Object Reference (IDOR) 23
8.1. Principe 23
8.2. Exemple d'attaque IDOR 24
8.3. Méthode traditionnelle de protection 24
9. Attaque par Remote Code Execution (RCE) 25
9.1. Principe 25

9.2. Méthodes traditionnelles de protection 25
10. Attaque par File inclusion 26
10.1. Attaque par Local file inclusion (LFI) 27
10.2. Attaque par Remote File Inclusion (RFI) 27
10.3. Méthode traditionnelle de protéction 27
11. Attaque par Path Traversal (Directory Traversal) 28
11.1. Principe 28
11.2. Exemple d'attaque Path Traversal 28
11.3. Méthodes traditionnelles de protection 29
12. Attaque par Command Injection 30
12.1. Principe 30
12.2. Exemple d'attaque Command Injection 31
12.3. Méthode traditionnelle de protection 32
Conclusion 32
Chapitre 2 : L'apprentissage automatique dans la sécurité des applications web 33
Introduction 34
1. Les dataset les plus traités dans les études antérieures 34
1.1. Le dataset KDD-Cup 99 34
1.2. Le dataset NSL-KDD 36
1.3. Le dataset ECML/PKDD 2007 37
1.4. Le dataset CSIC HTTP 2010 38
1.5. Le dataset UNSW-NB15 40
1.6. Le dataset HttpParams 41
1.7. Le dataset CIC-IDS 2017 42
1.8. Le dataset CSE-CIC-IDS2018 43
1.9. Le dataset Hybrid (CSIC_HTTPParams) 45
1.10. Le dataset SR-BH 2020 46
2. Comparaison entre les datasets 52
3. Choix des datasets pour application de modèles 56
Chapitre 3 : La conception d'un système De détection basé sur l'apprentissage automatique 57
Introduction 59
1. Outils et environnement de développement 59
2. Prétraitement des données 59
2.1. Nettoyage des données (Data cleaning) 59
2.2. Transformation des données (Data transformation) 60
2.3. Sélection des caractéristiques (Feature selection) 60
2.4. Gestion des données déséquilibrées 60
2.5. Division du dataset (Data splitting) 60
3. Prétraitement des données de dataset (CSIC_HTTPParams) 61
3.1. Nettoyage des données (Data cleaning) 61
3.1.1. Vérification des types de données 61
3.1.2. Traitement des valeurs manquantes 61
3.1.3. Analyse des incohérences 61
3.1.4. Suppression des doublons 62
3.2. Sélection des caractéristiques (Feature selection) 62
3.3. Division du dataset (Data splitting) 62
3.4. Gestion des données déséquilibrées 62
3.5. Transformation des données (Data transformation) 64
4. Prétraitement des données de dataset SR-BH 2020 64

4.1. Nettoyage des données (Data cleaning) 65
4.1.1. La suppression de doublons 66
4.1.2. Filtrage de la variable cible 66
4.2. Sélection des caractéristique (Feature selection) 66
4.3. Transformation des données (Data transformation) 67
4.4. Division du dataset (Data splitting) 67
4.5. Gestion des données déséquilibrées 68
5. Application des modèles d'apprentissage automatique sur le dataset 69
5.1. Choix des modèles 69
5.2 Mesures d'évaluation des Modèles de Classification 70
5.3. Application des modèles 72
5.3.1 Random Forest 72
5.3.2 Régression logistique 75
5.3.3 LightGBM 79
5.3.4 XGBoost(eXtreme Gradient Boosting) 82
5.4. Comparaison entre les résultats obtenus de classification binaire 85
5.5. Comparaison des résultats obtenus sur dataset CSIC_HTTPParams et les études antérieures 86
5.6. Comparaison entre les résultats des différents modèles de classification multiclasse 87
5.7. Comparaison des résultats du dataset SR-BH 2020 avec les études antérieures 89
Conclusion 92
Chapitre 4 : Déploiement du modèle dans un environnement réel 89
Introduction 94
1. Environnement de test 94
1.1. Mise en place de l'environnement expérimental local 94
1.2. Configuration du serveur web Apache2 95
1.3. Présentation et exploitation du fichier access.log 95
1.4. Installation des dépendances et du modèle de détection 96
2. Architecture du système de détection 97
3. Pipeline de traitement en temps réel 99
3.1. Lecture continue du fichier access.log 99
3.2 Extraction des caractéristiques (features) 99
3.3 Passage au modèle pour prédiction 100
3.4. Journalisation et alerte en cas d'attaque détectée 100
4. Exemple de fonctionnement 101
4.1. Génération de la requête malveillante 101
4.2 Analyse de la requête via le script Python 101
4.3. Prédiction par le modèle 101
4.4. Journalisation et capture 102
5. Mise à l'épreuve du système de détection choisi dans différents scénarios 103
Conclusion 105
Conclusion générale 107
Références XVII

Liste des figures

Figure 1: Exemple de SQL injection 6
Figure 2 : DOS et DDOS 9
Figure 3 : Principe d'une attaque XSS persistant 12
Figure 4: Principe d'une attaque XSS non persistant 13
Figure 5: Principe d'une attaque MitM 16
Figure 6:Exemple d'attaque Credential Stuffing 18
Figure 7: Exemple d'attaque Brute Force 19
Figure 8:Exemple d'attaque XEE 22
Figure 9: une attaque IDOR par YouTube 24
Figure 10: Processus de RCE 25
Figure 11 : Exemple d'attaque Path Traversal 29
Figure 12 : Exemple d'attaque Command Injection 31
Figure 13: Pourcentage des attaques et non attaques dans le dataset SR-BH 2020 dataset 46
Figure 14 : Distribution des lignes vide dans le dataset 49
Figure 15:distribution de type d'attaque dans le dataset 50
Figure 16 : Répartition des attaques par type sur les requêtes dans le dataset 51
Figure 17:Pourcentage des attaques dans le dataset 52
Figure 18 : Distribution des classes dans le training set CSIC_HTTPParams(avant SMOTE) 63
Figure 19 : Distribution des classes dans le training set CSIC_HTTPParams(après SMOTE) 64
Figure 20 : Répartition en pourcentage des données multi-label et multi-classe 65
Figure 21 : L'importance des caractéristiques selon le test ANOVA 67
Figure 22 : Distribution des classes avant l'application de SMOTE 68
Figure 23 : Distribution des classes après l'application de SMOTE 69
Figure 24 : Matrice de confusion Random Forest en classification binaire 73
Figure 25 : Matrice de confusion Random Forest en classification multiclass 75
Figure 26 : Matrice de confusion Regression Logistique en classification binaire 77
Figure 27 : Matrice de confusion Regression Logistique en classification multiclasse 78
Figure 28 : Matrice de confusion LightGBM en classification binaire 80
Figure 29 : Matrice de confusion LightGBM en classification multiclasse 81
Figure 30 : Matrice de confusion XGBoost en classification binaire 83
Figure 31 : Matrice de confusion XGBoost en classification multiclasse 84

Figure 32: Résultats d'application de modèles sur dataset Hybrid 86
Figure 33 : Résultats obtenus par shaheed et Kurdy 2022, sur le dataset Hybrid 87
Figure 34 : Comparaison du modèle LightGBM dans le présent travail et dans (Sureda Riera et al, 2022) 90
Figure 35 : Comparaison entre les deux modèles à mesures plus enlevés dans les deux travaux91
Figure 36 : Schéma d'analyse de la sécurité des applications web 98

Liste des tableaux

Tableau 1: Différents types de trafic dans dataset KDD Cup 99 35
Tableau 2: Distribution des requêtes dans NSL-KDD 36
Tableau 3: Distribution des requêtes dans ECML/PKDD 2007 37
Tableau 4: Analyse des caractéristiques (features) de dataset CSIC HTTP 2010 39
Tableau 5: Distribution des requêtes dans UNSW-NB15 40
Tableau 6 : Distribution des requêtes dans HttpParams 41
Tableau 7: Distribution des requêtes dans CIC-IDS2017 42
Tableau 8: Distribution des requêtes dans CSE-CIC-IDS2018 44
Tableau 9: Distribution des requêtes dans CSIC_HTTPParams 46
Tableau 10: Distribution des requêtes dans SR-BH 2020 47
Tableau 11 : Comparaison entre les datasets. 55
Tableau 12 : Éléments de la matrice de confusion 71
Tableau 13: Résultats de modèle Random Forest en classification binaire 72
Tableau 14 : Résultats du modèle RandomForest pour la classification multiclasse 74
Tableau 15: Résultats de modèle Logistic Regression en classification binaire 76
Tableau 16 : Résultats de modèle Logistic Regression en classification multiclasse 77
Tableau 17 : Résultats du modèle LightGBM pour la classification binaire 79
Tableau 18 : Résultats du modèle LightGBM pour la classification multiclasse 81
Tableau 19 : Résultats de modèle XGBoost en classification binaire 82
Tableau 20: Résultats de modèle XGBoost en classification multiclasse 84
Tableau 21 : Tableau comparatif des performances des modèles de classification binaire pour la détection des attaques 85
Tableau 22 : Tableau comparatif des performances des modèles de classification multiclasse pour la détection des attaques 88
Tableau 23 : Résultats des mesures du modèle LightGBM dans les deux travaux 90
Tableau 24 : Les meilleurs résultats obtenus sur les deux travaux 91
Tableau 25 : Signification des champs dans le fichier access.log 96
Tableau 26 : Résultats de la détection pour différents types de requêtes HTTP 103

Abréviations

- **ACCS : Centre Australien pour la Cybersécurité**
- **API : Application Programming Interface**
- **ASCII : American Standard Code for Information Interchange**
- **ASP : Active Server Pages**
- **CIC : Canadian Institute for Cybersecurity, : Canadian Institute for Cybersecurity**
- **CLF : Common Log Format**
- **CN : Common Name**
- **CSE : Communication Security Establishment**
- **CSRF :Cross-Site Request Forgery**
- **DDoS : Distributed Denial of Service**
- **DoS : Denial of Service**
- **DTD :Document Type Definition**
- **ECML : European Conference on Machine Learning**
- **FAI : Fournisseur d'Accès à Internet**
- **HTML : HyperText Markup Language**
- **HTTP : Hypertext Transfer Protocol**
- **ICMP : Internet Control Message Protocol**
- **IDOR : Insecure Direct Object Reference**
- **IDS : Systèmes de détection d'intrusions**
- **IP :Internet Protocol**
- **JSP : Java Server Pages**
- **LAN : Local Area Network**
- **LFI : Local File Inclusion**

- **LIRMM : Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier**
- **MFA : Multi factor authentification**
- **MIT : Massachusetts Institute of Technology**
- **MitM : Man-in-the-Middle**
- **OWASP : Open Worldwide Application Security Project**
- **PHP : Hypertext Preprocessor**
- **PKDD : Principles and Practice of Knowledge Discovery in Databases**
- **R2L : Remote to Local**
- **RCE : Remote Code Execution**
- **RFI : Remote File Inclusion**
- **SMOTE : Synthetic Minority Over-sampling Technique**
- **SSL/TLS : Secure Sockets Layer/Transport Layer Security**
- **SYN /ACK : Synchronization-Acknowledgement**
- **TCP : Transmission Control Protoco**
- **U2R : User to Root**
- **UDP : User Datagram Protocol**
- **UNB : University of New Brunswick**
- **UNSW : l'Université de New South Wales**
- **URL : Uniform Resource Locator**
- **WAF : Web Application Firewall**
- **WSL : Windows Subsystem for Linux**
- **XEE : XML External Entity**
- **XML :Extensible Markup Language**
- **XSS : Cross-Site Scripting**

Introduction générale

Introduction générale

Avec la diffusion considérable d'Internet et la généralisation des applications web dans les domaines économiques, sociaux et administratifs, la sécurité de ces plateformes est devenue un enjeu stratégique majeur. En effet, les applications web sont aujourd'hui la cible d'un large éventail d'attaques, allant des injections SQL et du Cross-Site Scripting (XSS), jusqu'aux attaques par déni de service (DDoS) et à l'exécution de code à distance (RCE). Ces attaques, de plus en plus complexes et fréquentes, exploitent des vulnérabilités souvent liées à une mauvaise validation des entrées ou à des défauts de configuration.

Face aux limites des solutions de sécurité classiques telles que les pare-feux applicatifs statiques, les systèmes de filtrage basiques ou encore la supervision manuelle, de nouvelles approches basées sur l'intelligence artificielle ont émergé. En particulier, l'apprentissage automatique offre des capacités puissantes de détection d'intrusions en s'appuyant sur l'analyse de comportements anormaux à partir de grands volumes de données, tout en s'adaptant continuellement aux nouvelles formes d'attaques.

Dans ce cadre, notre travail vise à développer un système intelligent de détection des attaques web basé sur des techniques d'apprentissage automatique. Ce système repose sur plusieurs étapes essentielles : la sélection rigoureuse de datasets pertinents, récents et représentatifs des menaces actuelles ; le prétraitement avancé des données (nettoyage, encodage, sélection de caractéristiques) ; l'évaluation comparative de plusieurs modèles de classification (Random Forest, LightGBM, XGBoost, etc.) ; ainsi qu'une validation dans un environnement local simulant des conditions réelles d'utilisation.

Ce mémoire est organisé en quatre chapitres :

Le premier chapitre présentera les fondements théoriques de la sécurité des applications web, en explorant les types de vulnérabilités, les vecteurs d'attaques et les mécanismes de défense traditionnels.

Le deuxième chapitre sera consacré à l'introduction de différents datasets mentionnées dans les études antérieures. Nous allons par la suite choisir, parmi eux, quelques datasets sur lesquels on va appliquer des modèles d'apprentissage automatique.

Le troisième chapitre décrit en détail la méthodologie suivie, depuis le choix des datasets (CSIC_HTTPParams et SR-BH 2020) jusqu'au développement du système de détection et l'analyse des résultats expérimentaux.

Le quatrième chapitre s'intéresse à l'évaluation en environnement réel, à travers

l'intégration du système dans un serveur local Apache, et l'analyse des fichiers log pour tester la robustesse du modèle dans un contexte d'exécution pratique.

À travers cette étude, nous espérons contribuer à l'amélioration des mécanismes de protection des applications web, en proposant une approche intelligente, évolutive et adaptée aux menaces contemporaines, capable d'intégration dans des contextes réels et diversifiés.

Chapitre 1 : Généralités sur la sécurité des applications web

Introduction

Les applications web sont de plus en plus largement utilisées dans nos vies quotidiennes et professionnelles, mais cette croissance s'accompagne d'une exposition accrue à diverses menaces de sécurité. Certaines menaces peuvent avoir des conséquences graves, telles que le vol de données sensibles, la perturbation du bon fonctionnement des systèmes ou encore l'indisponibilité des services. Ces attaques exploitent souvent des vulnérabilités spécifiques dans le code ou la configuration des applications afin de compromettre leur intégrité, leur confidentialité ou leur disponibilité. Dans ce chapitre, nous mettons en lumière quelques attaques courantes qui représentent une menace réelle pour la sécurité des applications web, notamment les injections SQL, les attaques Cross-Site Scripting (XSS), le Cross-Site Request Forgery (CSRF), ainsi que les attaques par déni de service (DoS/DDoS). L'analyse de ces vecteurs d'attaque nous permettra de mieux comprendre les enjeux liés à la cybersécurité des environnements web.

1. Attaque par injection SQL (SQL injection)

1.1 Principe

L'attaque par injection est l'une des plus critiques en raison de sa grande diffusion. L'objectif principal de l'attaquant est de modifier la sémantique des requêtes SQL afin de manipuler ou de compromettre la base de données. Pour ce faire, l'attaquant introduit des caractères dangereux dans un champ d'entrée afin de modifier la structure d'une requête dynamique lors de son exécution. Les attaques par injection SQL peuvent exploiter trois mécanismes principaux. D'abord, l'injection dans les entrées d'utilisateur consiste à insérer du code SQL malveillant dans des champs de saisie non sécurisés d'une application Web. Ensuite, l'injection dans les cookies de connexion survient lorsque des données stockées en clair dans les cookies sont manipulées par un attaquant pour exécuter du SQL malveillant. Enfin, l'injection dans les variables du serveur exploite la manipulation des en-têtes Hypertext Transfer Protocol (HTTP) pour insérer des requêtes SQL nuisibles lors des échanges entre le serveur et la base de données. (Yassin, 2014)

1.2 Exemple d'attaque SQL injection

Un pirate profite d'une faille dans le formulaire de connexion de Facebook en entrant une requête SQL comme 'OR'1'='1' dans le champ utilisateur. Cette commande détourne la requête SQL d'authentification, ce qui permet d'accéder à un compte sans mot de passe. Ainsi, le pirate peut alors consulter ou modifier les données de l'utilisateur. Ce type d'attaque montre l'importance de sécuriser les champs de saisie à l'aide de requêtes préparées et de filtres de validation. La figure suivante illustre ce mécanisme d'injection SQL.

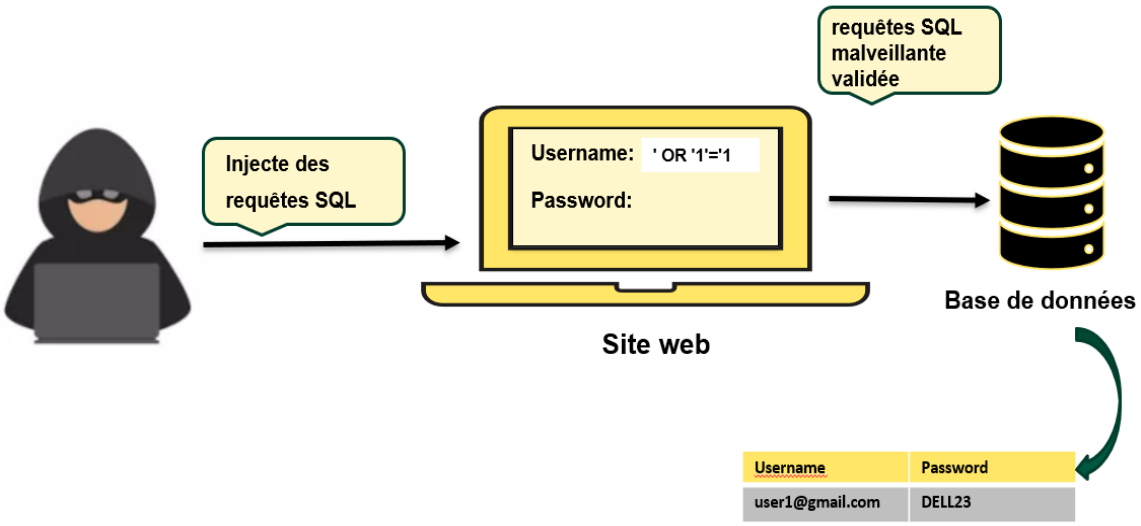

Figure 1: Exemple de SQL injection

1.3. Méthodes traditionnelles de protection

Pour se protéger des injections SQL, Nous allons examiner les solutions existantes suivantes: la validation et le filtrage des entrées pour contrôler les données saisies, ainsi que la limitation des privilèges pour réduire l'impact des attaques réussies.

- **Validation des entrées**

La validation insuffisante des entrées est la cause fondamentale des injections SQL. Une méthode de protection intuitive consiste donc à vérifier si ces entrées contiennent des caractères dangereux. En règle générale, chaque champ d'entrée est associé à une colonne d'une table de la base de données. Avant de générer des requêtes SQL dynamiques, il est recommandé de valider les données afin de s'assurer qu'elles correspondent bien au type de la colonne à laquelle elles sont associées. Toutefois, cette méthode n'est efficace que pour les données numériques,

car les caractères non numériques peuvent être rejetés. En revanche, elle ne permet pas de sécuriser les entrées sous forme de chaînes de caractères, qui constituent la principale source d'attaques par injection SQL. (Yassin, 2014)

- **Filtrage des entrées**

Cette méthode consiste à bloquer les caractères potentiellement dangereux (ex. ', -) et à autoriser uniquement les lettres et chiffres. Bien qu'elle limite les risques, elle génère un taux élevé de faux positifs et ne peut couvrir toutes les variantes d'attaques. (Yassin, 2014)

- **Limitation des privilèges**

Accorder aux utilisateurs uniquement les permissions strictement nécessaires réduit les impacts d'une attaque réussie. Cependant, cette mesure ne protège pas contre les menaces internes et les attaques exploitant des failles dans les champs de recherche.

Aucune de ces méthodes ne garantit une protection absolue contre les attaques par injection SQL. Cependant, en combinant la validation et le filtrage des entrées avec une gestion stricte des privilèges, il est possible de réduire significativement les risques. Pour une sécurité optimale, ces approches doivent être intégrées à d'autres mécanismes de protection, tels que l'utilisation de requêtes paramétrées et de pare-feu applicatifs. (Yassin, 2014)

2. Attaque par Cross Site Request Forgery (CSRF)

2.1. Principe

Une attaque CSRF exploite la confiance qu'une application Web vulnérable accorde au navigateur d'un utilisateur authentifié. Elle contraint ce dernier à envoyer, à son insu, une requête HTTP malveillante incluant son cookie de session.

Cette attaque repose sur la prévisibilité des structures de requêtes des applications Web. Étant donné que les cookies de session sont envoyés automatiquement avec chaque requête, un attaquant peut intégrer des éléments invisibles dans une page Web piégée afin de générer des requêtes frauduleuses, indiscernables des requêtes légitimes.

Pour exécuter l'attaque, l'attaquant fabrique une requête HTTP et incite la victime à la soumettre, par exemple via une balise d'image (<IMG>) ou d'autres méthodes détournées. Si l'utilisateur est authentifié au moment de l'envoi, la requête est exécutée avec succès,

permettant ainsi à l'attaquant d'effectuer des actions malveillantes à son insu. Cela peut inclure la modification de données appartenant à la victime ou l'exécution d'actions en son nom. (Akrout, 2012)

2.2. Exemple d'attaque CSRF

Une application vulnérable permet aux utilisateurs de transférer de l'argent via une requête HTTP sans protection CSRF, comme

```
http://example.com/app/transferFunds?amount=1500&destinationAccount
```

Un attaquant exploite cette faille en intégrant une requête frauduleuse dans une balise d'image invisible sur un site piégé :

```
<img
src="http://example.com/app/transferFunds?amount=1500&destinationAccoun
t=AttackerAccount" width="0" height="0">
```

Si une victime authentifiée visite ce site, son navigateur enverra automatiquement la requête avec ses informations de session, validant ainsi le transfert à son insu. Pour prévenir cette attaque, il est essentiel d'utiliser des tokens CSRF, de vérifier l'origine des requêtes et d'exiger une confirmation utilisateur pour les transactions sensibles.

2.3. Méthodes traditionnelles de protection

Il existe plusieurs stratégies couramment utilisées pour contrer les attaques CSRF. Nous examinerons ci-dessous les plus répandues.

- **Vérification de l'origine de la requête**

Cette méthode consiste à vérifier l'origine de la requête. Cependant, le seul fait de vérifier l'entête HTTP REFERRER n'est pas suffisant, car il peut être facilement falsifié. (Hossen, 2014)

- **Utilisation d'un jeton de sécurité**

Cette approche repose sur l'ajout d'une valeur aléatoire dans les paramètres de la requête, appelée jeton de sécurité. Grâce à son caractère aléatoire, il permet de détecter et de protéger contre les attaques CSRF**.** (Hossen, 2014)

Ainsi, l'association de ces deux méthodes constitue une approche essentielle pour sécuriser les applications web contre les attaques CSRF et garantir l'intégrité des requêtes.

3. Attaques par Déni de Service (DOS/DDOS)

3.1. Principe

Il existe deux types d'attaques par déni de service : le déni de service (DoS) et le déni de service distribué (DDoS). Elles visent à rendre un appareil, un réseau ou un service inaccessible. Ces attaques peuvent cibler la couche applicative en exploitant des failles logicielles ou en submergeant les ressources du serveur, ce qui limite l'accès au service pour les utilisateurs connectés et leur aptitude à l'utiliser efficacement. (Mammeri, 2024)

En revanche, une attaque Dos est menée par un seul individu ou système qui inonde le service ciblé avec des requêtes afin de perturber son fonctionnement. Pour éviter d'être identifié, un hacker peut masquer son identité réseau (adresse IP et ports UDP/TCP) en usurpant l'identité d'une autre machine (spoofing), ce qui rend la détection et la lutte contre l'attaque plus difficiles. La Figure 02 illustre ce type d'attaque.

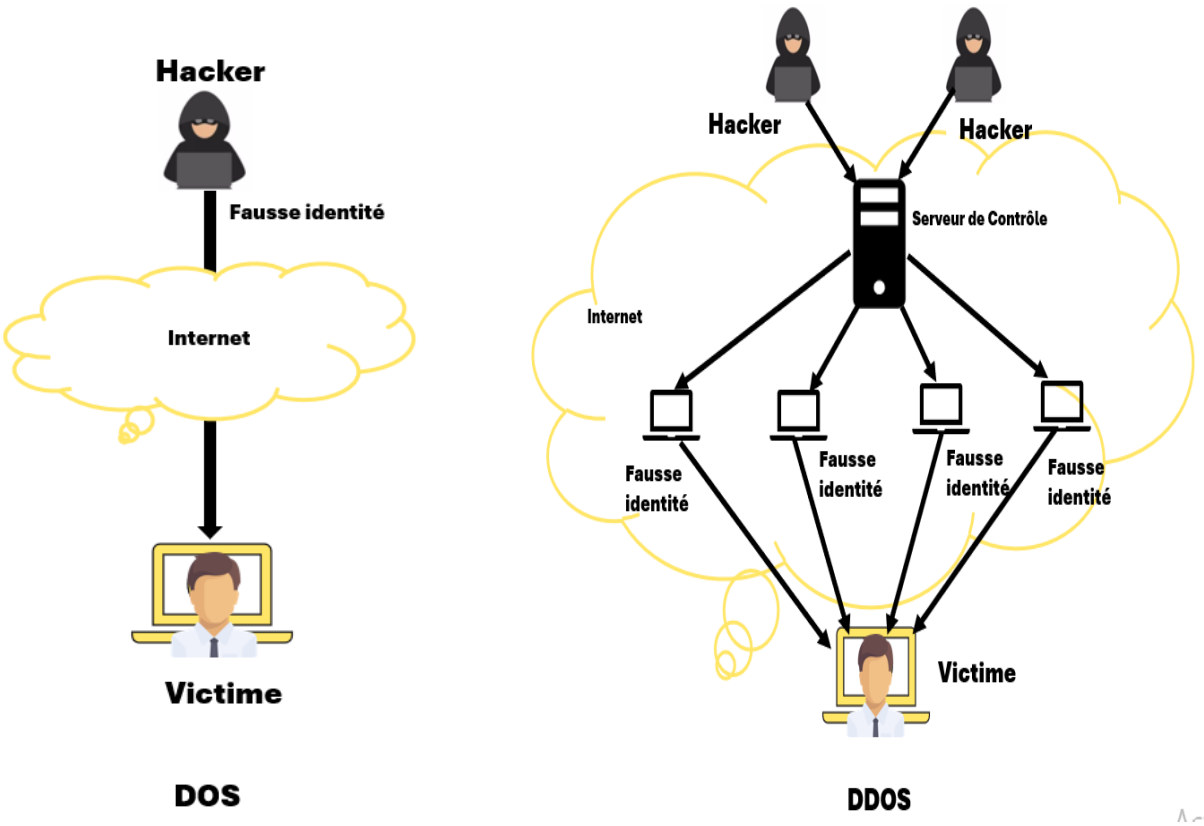

Figure 2 : **DOS et DDOS**

Le DDoS détient trois types généraux : les attaques volumétriques, les attaques de protocole et les attaques de la couche application, également appelées attaques DDoS de couche 7. Ces dernières ciblent spécifiquement la génération et la livraison des pages Web en exploitant des requêtes HTTP coûteuses à traiter pour le serveur. Elles incluent les inondations GET/POST, les attaques lentes et l'exploitation de vulnérabilités spécifiques. Difficiles à détecter en raison de leur apparence légitime, leur impact est mesuré en requêtes par seconde.

Les attaques de protocole saturent les tables d'état des équipements réseau (pare-feu, Load balancers) via des attaques TCP synchronize/acknowledge (SYN /ACK), et autres, mesurées en paquets par seconde. Les attaques volumétriques, quant à elles, visent à submerger la bande passante avec des inondations ICMP, UDP et autres, mesurées en bits par seconde. (RAZAFINDRAKOTO , 2023)

3.2. Méthodes traditionnelles de protection

Avec l'évolution de la technologie, diverses méthodes ont été mises en place pour se protéger spécifiquement contre les attaques DDoS en adoptant les mesures préventives suivantes.

- **Détection basée sur la détection d'anomalies statistiques**

Cette méthode de détection des attaques DDoS offre une protection efficace contre les attaques inconnues car elle se concentre sur la surveillance du trafic réseau afin d'identifier des anomalies et des schémas inhabituels indiquant une éventuelle menace. Cette approche repose sur l'élaboration de modèles statistiques du trafic réseau normal afin de détecter toute anomalie significative. En analysant les données historiques, un modèle est établi en tenant compte de divers paramètres comme le volume de trafic et la fréquence des requêtes. Le trafic en temps réel est ensuite comparé aux valeurs prévues, et toute divergence marquée est considérée comme une anomalie. Ces anomalies, telles que des pics soudains ou des connexions suspectes, déclenchent des alertes de sécurité, permettant une intervention rapide pour atténuer les menaces. (Boin, 2023)

- **Mettre en œuvre un routage par trou noir**

La mise en trou noir est une contre-mesure utilisée pour atténuer une attaque DDoS en redirigeant tout le trafic, légitime ou malveillant, vers une route nulle. Avec l'aide du

Fournisseur d'Accès à Internet (FAI), l'administrateur réseau peut configurer cette technique pour isoler une adresse IP ciblée. Toutefois, si les critères de filtrage ne sont pas précis, cette méthode peut engendrer des pertes commerciales en rendant le réseau inaccessible, atteignant ainsi l'objectif des attaquants. (Centre canadien pour la cybersécurité, 2024)

- **Appliquer la limitation de débit**

La limitation de débit est une solution efficace pour atténuer les attaques DDoS en restreignant le nombre de requêtes qu'un serveur peut accepter depuis une même adresse IP sur une période donnée. En réduisant le trafic excessif, elle empêche les attaquants de saturer les ressources du système, tout en maintenant l'accessibilité pour les utilisateurs légitimes. Bien qu'insuffisante contre des attaques DDoS sophistiquées, cette approche joue un rôle essentiel dans une stratégie de défense globale. (Centre canadien pour la cybersécurité, 2024)

4. Attaque par Cross-Site Scripting (XSS)

4.1. Principe

Les attaques XSS se produisent lorsque les applications web intègrent des données non fiables fournies par les utilisateurs sans les vérifier ou les nettoyer, avant de les renvoyer à d'autres utilisateurs sous forme de contenu web. Cela permet à un attaquant d'injecter du code malveillant qui sera exécuté dans le navigateur des victimes, ce qui peut entraîner la corruption du contenu d'un site ou rediriger la victime vers un autre site malveillant. (Hossen, 2014)
On distingue généralement deux types d'attaques XSS :

XSS persistant (Stored XSS) : Ce type d'attaque est considéré comme l'une des menaces de sécurité les plus dangereuses. Les cybercriminels injectent un code malveillant dans la base de données d'un site web. Par exemple, ils peuvent laisser des commentaires sur des sites d'actualités. La prochaine fois qu'un utilisateur visite le site et consulte les commentaires, le code s'exécutera automatiquement dans son navigateur à son insu. Cette attaque est particulièrement préoccupante, car le code malveillant reste stocké dans les données du site pendant une longue période, à moins qu'il ne soit détecté et éliminé, exposant ainsi un large éventail de victimes à des risques fréquents, comme illustré dans la figure suivante.

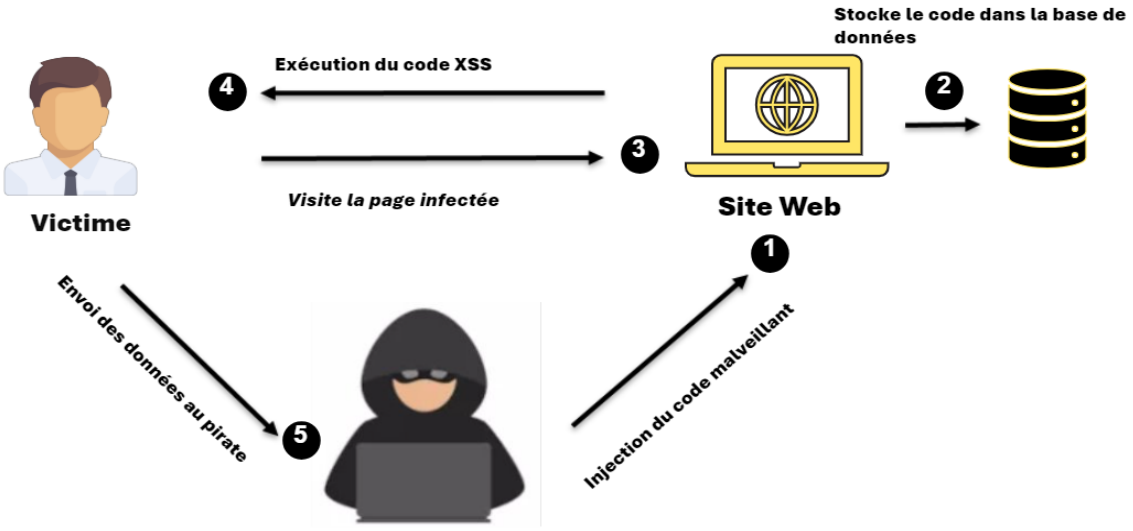

Figure 3 : Principe d'une attaque XSS persistant

XSS non persistant (Reflected XSS) : Le XSS non persistant n'implique pas de stocker le script malveillant sur le serveur, mais plutôt de l'envoyer par une requête particulière. L'attaquant modifie un paramètre d'Uniform Resource Locator (URL) ou injecte du code dans un champ de formulaire, et trompe la victime visée pour cliquer sur un lien illégitime, qui arrive généralement par courrier électronique ou est intégré dans une page Web apparemment fiable. Lorsque ces données sont soumises par l'utilisateur dans une requête, l'application Web renvoie simplement les données saisies sans échappement dans le page web de résultats, ce qui entraîne l'exécution du script dans le navigateur de l'utilisateur.

De telles attaques sont relativement courantes avec les systèmes de gestion de contenu car les liens peuvent être partagés et être visibles par un grand nombre d'utilisateurs Internet, comme le démontre la figure suivante.

Figure 4: Principe d'une attaque XSS non persistant

4.2. Exemples d'attaques XSS

4.2.1. Attaque XSS persistant

L'attaque XSS persistante consiste à injecter un code malveillant dans un site Web vulnérable, où il est stocké dans une base de données et affiché automatiquement à chaque utilisateur visitant la page compromise, sans nécessiter d'interaction de leur part.

Par exemple, un **forum** permettant aux utilisateurs de publier des commentaires sans filtrage adéquat peut être exploité. Un attaquant pourrait insérer le code malveillant suivant :

```
<script>document.write('<img src="http://site-attaquant.com/steal.php?
cookie=' + document.cookie + '">');</script>
```

Lorsque ce commentaire est enregistré dans la base de données et affiché aux autres utilisateurs, le script s'exécute automatiquement dans leur navigateur.

4.2.2. Attaque XSS non persistant

L'attaque XSS non persistante consiste à injecter un code malveillant dans un site Web vulnérable sans qu'il soit stocké dans une base de données. Le code s'exécute uniquement

lorsque l'utilisateur clique sur un lien piégé contenant la charge malveillante.

Par exemple, un attaquant peut exploiter un service de messagerie Web (Webmail) en envoyant un e-mail contenant un lien malveillant :

Cliquez ici pour voir un message important :

```
<a href="http://www.site-vulnerable.com/message.php?msg=<script>alert('XSS
Non Persistant');</script>">Lire le message</a>
```

Lorsque la victime ouvre l'e-mail et clique sur le lien, le script s'exécute immédiatement dans son navigateur.

4.3. Méthode traditionnelle de protection

La prévention des attaques XSS repose principalement sur une gestion rigoureuse des flux de données entre l'utilisateur et l'application. La première étape consiste à filtrer les entrées utilisateur afin de détecter et neutraliser tout contenu HyperText Markup Language (HTML) ou JavaScript potentiellement malveillant. Il est essentiel d'échapper ou d'encoder les caractères spéciaux susceptibles d'être interprétés comme du code exécutable, notamment les caractères suivants (Guillaume, 2012):

```
& &amp;
< → &lt;
> → &gt;
" → &quot;
' → &#x27;  (' n'est pas recommendé)
/ → &#x2F;
```

L'encodage doit être adapté au contexte dans lequel les données sont affichées : HTML, JavaScript, ou URL. Par exemple, la fonction htmlspecialchars () en Hypertext Preprocessor (PHP) permet de transformer des balises HTML (<script>) en texte inoffensif (<script>), empêchant ainsi leur interprétation par le navigateur. (Guillaume, 2012)

Il est également recommandé d'utiliser des listes blanches pour valider les formats acceptés, comme dans le cas d'un champ de numéro de téléphone qui ne doit contenir que des chiffres. Du côté client, JavaScript doit valider les données saisies, tandis que du côté serveur, il faut impérativement contrôler les paramètres reçus et rejeter toute donnée non conforme.

Pour se prémunir contre le vol de cookies via JavaScript, l'utilisation de l'attribut HTTPOnly est fortement conseillée. Celui-ci empêche tout accès aux cookies via les scripts, comme illustré dans l'exemple PHP suivant (Guillaume, 2012) :

```
<?php session.cookie_httponly = True?>
```

Cependant, les solutions traditionnelles telles que l'encodage des caractères spéciaux et la validation des entrées ne sont pas toujours suffisantes pour se défendre contre les attaques XSS. Par exemple, la validation des entrées peut être contournée dans certains cas, tandis que l'attribut HTTPOnly reste limité à la protection des cookies sans offrir une protection complète contre toutes les menaces.

5. Attaque par Man-in-the-Middle (MitM)

5.1. Principe

L'attaque Man-in-the-Middle (MitM), ou "l'homme du milieu", consiste à intercepter et manipuler les communications entre deux systèmes sans que ces derniers ne s'en rendent compte. Dans une transaction HTTP, par exemple, la cible est souvent la connexion TCP entre le client et le serveur. L'attaquant interpose son système entre les deux parties en divisant la connexion TCP originale en deux nouvelles connexions : l'une entre le client et l'attaquant, et l'autre entre l'attaquant et le serveur. Agissant comme un proxy, il peut ainsi lire, modifier ou insérer des données dans les échanges interceptés.

Cette attaque est particulièrement efficace dans le contexte du protocole HTTP, car les données échangées sont généralement en American Standard Code for Information Interchange (ASCII), rendant leur lecture et manipulation plus aisée. Il devient alors possible, par exemple, de capturer un cookie de session à partir de l'en-tête HTTP ou encore de modifier une transaction financière dans le contenu applicatif.

Même les connexions HyperText Transfer Protocol Secure (HTTPS) ne sont pas totalement à l'abri. Dans ce cas, l'attaquant établit deux connexions Secure Sockets Layer/Transport Layer Security (SSL/TLS) indépendantes : une entre le navigateur et lui-même, et une autre entre lui et le serveur. Bien que le navigateur puisse afficher un avertissement concernant un certificat SSL invalide, l'utilisateur peut l'ignorer par méconnaissance du risque. Pire encore, aucun avertissement ne s'affiche si l'attaquant utilise

un certificat compromis ou émis par une autorité de certification de confiance avec un nom commun, (Common Name -CN) identique à celui du vrai site.

L'attaque MitM repose sur des techniques telles que l'usurpation d'IP, le détournement Domain Name System (DNS), la falsification de protocoles HTTPS, les attaques par e-mail, l'écoute de réseaux Wi-Fi non sécurisés ou encore les attaques par rétrogradation SSL/TLS. Elle permet d'accéder à des données sensibles comme les identifiants de connexion, les informations bancaires ou les détails de comptes personnels. Ces données peuvent ensuite être exploitées pour réaliser d'autres attaques, telles que l'usurpation d'identité, le piratage de comptes ou des transactions financières non autorisées.

Enfin, il convient de noter que la technique MitM n'est pas uniquement utilisée à des fins malveillantes : elle peut aussi être déployée dans un cadre légal lors du développement d'applications web ou lors d'audits de sécurité pour identifier des vulnérabilités dans les systèmes de communication, comme l'illustre la figure suivante. (OWASP Foundation, n.d.)

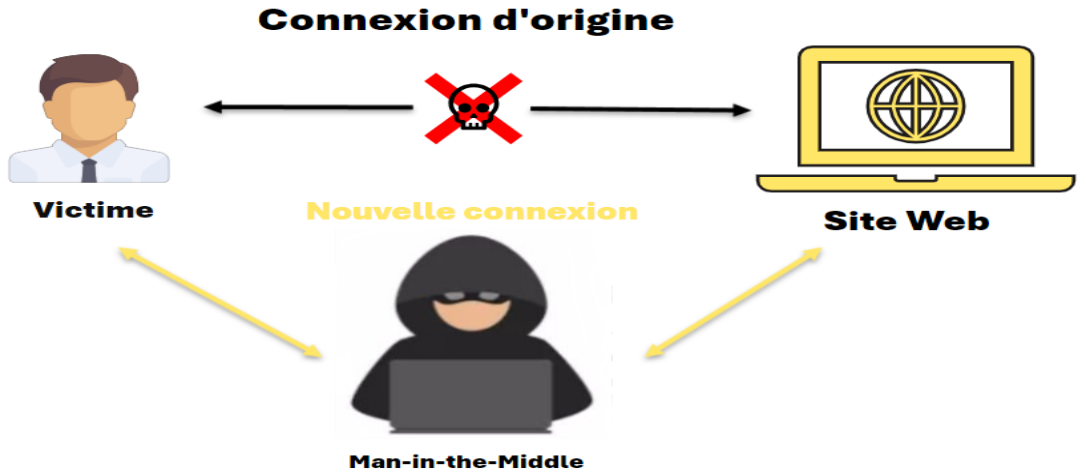

Figure 5: Principe d'une attaque MitM

5.2. Exemple d'attaque MitM

Un utilisateur reçoit un e-mail semblant provenir de son service de paiement en ligne (ex. PayPal). L'e-mail l'informe d'une activité suspecte sur son compte et l'invite à cliquer sur un lien pour vérifier ses informations.

Lorsqu'il clique sur le lien, il est redirigé vers un site qui ressemble exactement au vrai site de PayPal. Sans se méfier, il saisit ses identifiants et valide la connexion. En réalité, ce site est contrôlé par un attaquant qui intercepte les identifiants de l'utilisateur et les utilise immédiatement pour se connecter au vrai site PayPal et effectuer des transactions frauduleuses.

5.3. Méthodes traditionnelles de protection

- **Sécurisation des échanges par chiffrement**

En optant pour HTTPS plutôt que HTTP afin d'assurer la confidentialité des échanges. Cependant, même en mettant en œuvre cette règle, une protection totale n'est pas garantie, surtout dans le cas où un attaquant utilise un faux certificat (attaque SSL Strip). (BOUGHAZI & LAKHAL).

- **Pare-feu et antivirus**

L'utilisation d'antivirus et de pare-feu est indispensable pour se prémunir contre les attaques de type MitM. Le pare-feu inspecte le trafic réseau, identifie les actions douteuses et les empêche de parvenir à l'utilisateur. En outre, les programmes antivirus identifient et éliminent les logiciels malveillants susceptibles d'être employés pour surveiller les communications. Toutefois, si l'utilisateur consent à aller sur un site malveillant, le pare-feu ne sera pas capable d'empêcher l'attaque. De plus, les programmes antivirus demeurent peu efficaces face aux menaces non identifiées ou aux attaques sophistiquées qui réussissent à déjouer les bases de données de signatures classiques.

- **Renforcement de la sécurité par la mise à jour et la gestion des cookies**

L'actualisation fréquente des navigateurs, des systèmes d'exploitation et des logiciels de sécurité contribue à rectifier les failles que peuvent exploiter les attaquants. Par ailleurs, la désactivation de l'enregistrement automatique des cookies empêche le vol de session par des attaques telles que le détournement ou l'empoisonnement du cache. Néanmoins, malgré des actualisations régulières, de nouvelles vulnérabilités peuvent survenir, et la désactivation des cookies pourrait nuire à l'expérience utilisateur en nécessitant des authentifications répétées.

6. Attaque par Brute Force et Credential Stuffing

6.1. Principe

Les attaques par force brute et les attaques par credential stuffing sont deux types d'attaques visant le même objectif : obtenir un accès non autorisé aux comptes en exploitant les mots de passe des utilisateurs. Cependant, chaque type d'attaque repose sur une méthode

différente pour découvrir le mot de passe.

Attaque par Credential Stuffing **:** Cette attaque repose sur l'exploitation des informations de connexion (noms d'utilisateur et mots de passe) obtenues illégalement, que ce soit par des piratages ou des fuites de données. L'attaquant utilise ces informations pour les tester sur divers sites web et services, en supposant que certains utilisateurs réutilisent les mêmes identifiants sur plusieurs plateformes. Ainsi, si un compte a été compromis, l'attaquant peut potentiellement accéder à d'autres comptes utilisant les mêmes informations de connexion.

Attaque par force brute (Brute Force) **:** Dans ce type d'attaque, l'attaquant tente de forcer l'accès à un compte en testant systématiquement toutes les combinaisons possibles d'un mot de passe jusqu'à trouver la bonne. Bien que ce procédé semble simple, il est particulièrement long et exigeant en raison du grand nombre de possibilités, surtout lorsque les mots de passe sont complexes et longs.

6.2. Exemples d'attaques Brute Force et Credential Stuffing

6.2.1. Attaque Credential Stuffing

Un attaquant utilise des identifiants divulgués d'un site web piraté, comme Snapchat.com, pour voler les adresses Email et les mots de passe des utilisateurs. Il tente ensuite de les utiliser sur différentes plateformes comme Gmail, Facebook, Netflix et Amazon. Si un utilisateur utilise le même mot de passe sur différentes plateformes, ses comptes peuvent être facilement piratés sans avoir besoin de tester plusieurs mots de passe. Ce scénario d'attaque basé sur la réutilisation d'identifiants est représenté dans la figure suivante

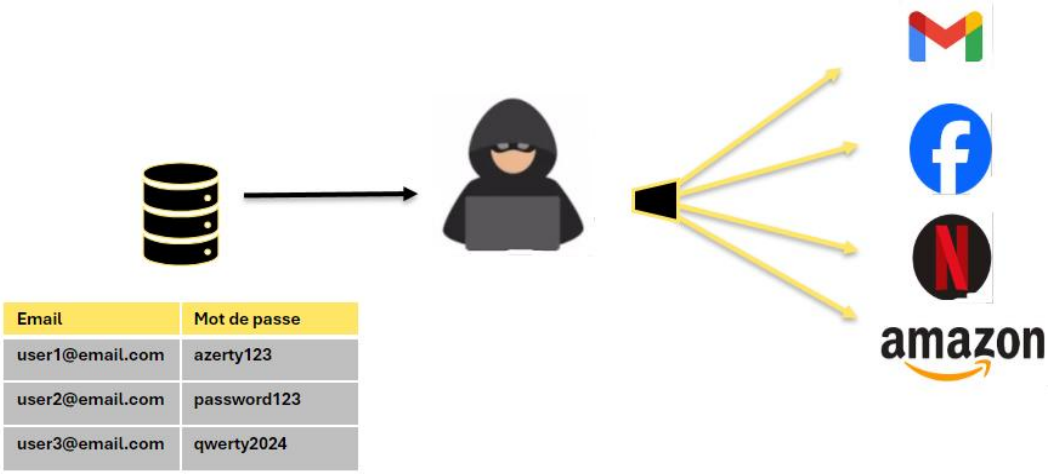

Figure 6:Exemple d'attaque Credential Stuffing

6.2.2. Attaque Brute Force

Un attaquant veut accéder à un compte bancaire en ligne qui nécessite un mot de passe à 6 caractères pour se connecter. Il ne connaît pas le mot de passe, alors il utilise une attaque par force brute pour tester toutes les combinaisons possibles jusqu'à en trouver la bonne. Le fonctionnement de ce type d'attaque est illustré dans la figure suivante.

Figure 7: Exemple d'attaque Brute Force

6.3. Méthodes traditionnelles de protection

Les attaques par force brute dépendent principalement de la faiblesse des mots de passe, c'est pourquoi la protection des données de connexion constitue la principale défense contre ces attaques. Voici quelques solutions traditionnelles à envisager pour renforcer la sécurité :

- **Utilisation des mots de passe uniques pour chaque compte**

Il est essentiel d'avoir un mot de passe distinct pour chaque compte afin d'éviter que la compromission des données d'un compte n'entraîne l'accès à d'autres comptes. Cette distribution des mots de passe renforce la sécurité et réduit l'impact d'une éventuelle violation. Cependant, de nombreux utilisateurs rencontrent des difficultés pour se souvenir de multiples mots de passe, ce qui peut les amener à utiliser des mots de passe faibles ou similaires à travers différents comptes.

- **Limitation le nombre de tentatives de connexion ou utiliser des techniques de temporisation**

En imposant des restrictions sur le nombre de tentatives successives de connexion ou en utilisant des techniques de temporisation entre les tentatives, on peut réduire l'efficacité des attaques automatisées. Ces restrictions rendent les attaques par force brute plus difficiles à exécuter et ralentissent leur progression. Cependant, cela peut entraîner une certaine gêne pour les utilisateurs qui oublient leur mot de passe ou commettent des erreurs de saisie, ce qui peut nuire à leur expérience.

- **Utilisation des mots de passe difficiles à deviner grâce à leur longueur et leur complexité**

Les mots de passe doivent être longs et complexes, incluant des majuscules, des minuscules, des chiffres et des caractères spéciaux, afin de compliquer les tentatives de devinette. Éviter les mots simples courants renforce la résistance des mots de passe aux attaques par force brute. Cependant, les utilisateurs peuvent rencontrer des difficultés pour se souvenir de mots de passe complexes, ce qui peut les amener à les stocker de manière non sécurisée ou à utiliser des mots de passe réutilisés.

- **Utilisation l'authentification multi-facteurs**

En ajoutant une couche supplémentaire de vérification, l'accès non autorisé devient plus difficile, même si le mot de passe est compromis. L'authentification multi-facteurs (Multi factor authentification -MFA) consiste à utiliser plusieurs moyens pour vérifier l'identité de l'utilisateur, tels que l'envoi d'un code de vérification par e-mail ou par téléphone mobile. Cette méthode renforce considérablement la sécurité en ajoutant une barrière supplémentaire contre les attaques, même si le mot de passe a été volé ou deviné avec succès. Toutefois, les utilisateurs peuvent rencontrer certaines difficultés à utiliser cette technique, surtout s'ils n'ont pas un accès permanent à l'appareil ou à l'application utilisée pour la vérification.

7.Attaque par XML External Entity (XEE)

7.1. Principe

Extensible Markup Language (XML) est un langage de balisage extensible créé pour la conservation et la transmission d'informations. Il établit des critères pour structurer les documents dans un format aisément compréhensible tant pour les humains que pour les machines.

Un attaquant pourrait interférer avec l'analyse des données XML d'une application en exploitant une faiblesse dans le traitement des documents XML. Cette attaque se produit lorsqu'un analyseur XML mal protégé ou mal paramétré traite une entrée XML contenant des références à des entités externes. Cela pourrait permettre à un attaquant d'accéder aux documents sur le serveur et d'interagir avec les systèmes back-end ou externes auxquels l'application a accès. Les conséquences potentielles incluent la divulgation de données sensibles, le déni de service, l'analyse des ports de l'appareil hébergeant l'analyseur et d'autres impacts sur le système. (MEGHZILI, 2023)

Pendant l'analyse, le parseur XML examine l'entité « xxe » et essaie d'accéder à son contenu. Quand l'entité est identifiée, elle révèle le contenu du fichier, permettant ainsi à un attaquant de l'intégrer dans le XML afin d'accéder à des informations qui ne devraient normalement pas être accessibles. L'analyseur commence par lire le fichier local :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:////etc/passwd" >]>
<xmlroot><xmlEntry>&xxe;3</xmlEntry></xmlroot>
```

Puis exécute l'appel HTTP vers le serveur distant :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "http://api.geonames.org/timezoneJSON"
>]>
<xmlroot><xmlEntry>&xxe;3</xmlEntry></xmlroot>
```

7.2. Exemple d'attaque XEE

Dans une attaque de type XXE, un attaquant injecte une entité externe malveillante dans un fichier XML traité par le serveur sans validation adéquate. Le fichier XML contient une entité qui pointe vers une ressource sensible du système, comme le fichier /etc/passwd. Lors de l'analyse du fichier, le parseur XML remplace l'entité par le contenu de ce fichier, exposant ainsi des données sensibles à l'attaquant via l'interface web.

Voici un exemple de code XML utilisé dans ce type d'attaque :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Lorsque ce fichier est traité, la balise &xxe ; est remplacée par le contenu du fichier /etc/passwd, ce qui permet à l'attaquant de lire des informations critiques du système. Cette vulnérabilité est due à l'absence de configuration sécurisée du parseur XML qui devrait empêcher l'accès aux entités externes. Le mécanisme de cette attaque est illustré à la figure 8.

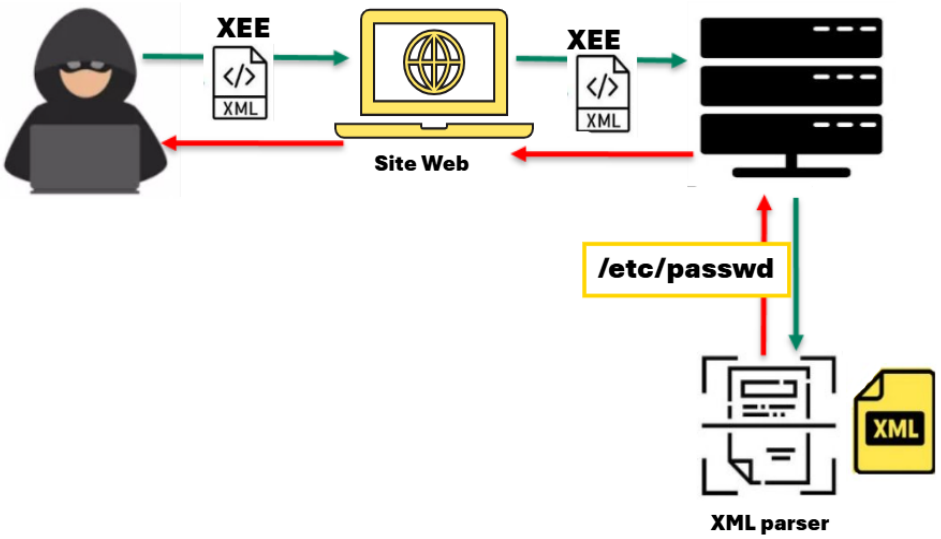

Figure 8:Exemple d'attaque XEE

7.3. Méthode traditionnelle de protection

Les solutions traditionnelles contre les attaques XXE incluent plusieurs bonnes pratiques pour sécuriser les analyseurs XML et éviter l'exploitation de vulnérabilités. Selon la feuille de triche Open Worldwide Application Security Project (OWASP) sur la prévention des attaques XXE, il est essentiel de désactiver les entités externes XML et le traitement de Document Type Definition (DTD) dans tous les analyseurs XML utilisés au sein de l'application. Cela empêche l'application de récupérer des données externes malveillantes via des entités XML, réduisant ainsi le risque d'injections malveillantes. De plus, il est recommandé de ne pas refléter le contenu XML directement à l'utilisateur, afin d'éviter d'exposer des informations sensibles ou des failles potentielles. Enfin, il est crucial de désactiver la récupération DTD externe, ce qui empêche l'inclusion de définitions externes pouvant être utilisées pour mener des attaques comme le vol de données ou l'exécution de commandes malicieuses. Ces mesures contribuent à protéger l'application contre les vulnérabilités XXE en limitant les interactions dangereuses avec le XML.

Cependant, bien que la désactivation des entités externes XML et du traitement DTD soit essentielle, ces pratiques ne garantissent pas une protection totale contre les attaques XXE, notamment si elles sont mal configurées ou si des bibliothèques obsolètes sont utilisées. De plus, ces mesures peuvent limiter certaines fonctionnalités légitimes des applications (MEGHZILI, 2023).

8. Attaque par Insecure Direct Object Reference (IDOR)

8.1. Principe

Les références non sécurisées à un objet, souvent directes, représentent une faille courante et importante dans les applications web. Cette vulnérabilité se manifeste lorsque l'application ne contrôle pas adéquatement les autorisations d'accès avant de traiter une demande. On peut retrouver ces références dans l'URL, les champs de saisie de formulaire, les en-têtes HTTP ou encore dans les cookies.

Une personne malintentionnée exploitant une vulnérabilité IDOR peut avoir accès à des données sensibles (telles que les dossiers de clients, les informations financières ou les documents internes), modifier des ressources cruciales ou usurper des comptes d'utilisateur.

8.2. Exemple d'attaque IDOR

Une vulnérabilité de type références directes non sécurisées (IDOR) a été identifiée sur YouTube en 2015 par le chercheur en sécurité russe Kamil Hismatullin. Cette faille permettait de supprimer n'importe quelle vidéo en modifiant simplement son identifiant lors d'une requête de suppression, sans qu'aucune vérification des droits d'accès ne soit effectuée. Représenté à la figure 9. (Jérôme & Jérôme , 2017)

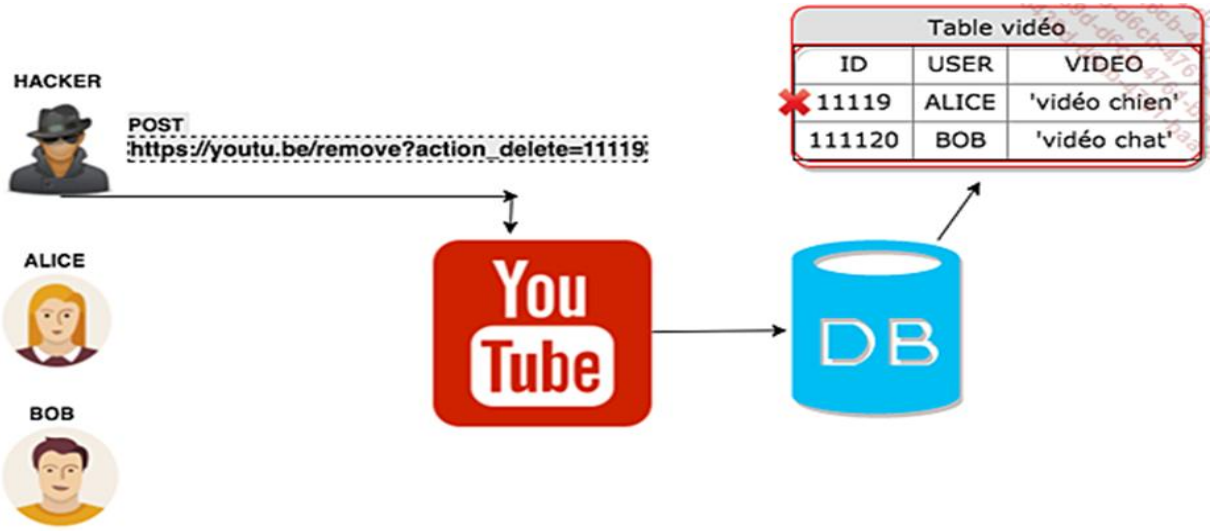

Figure 9: une attaque IDOR par YouTube

8.3. Méthode traditionnelle de protection

Pour réduire les dangers associés aux références directes non protégées (IDOR), l'instauration de dispositifs de contrôle d'accès stricts est indispensable. Il est courant de demander une nouvelle authentification de l'utilisateur avant d'accéder à des données sensibles ou à des fonctions essentielles. Ce contrôle additionnel garantit que la demande provient effectivement d'un utilisateur autorisé et légitime. (Guillaume, 2012)

Par exemple, plutôt que de se fier uniquement à un identifiant transmis dans la requête pour supprimer une vidéo, le système devrait s'assurer que l'utilisateur connecté est réellement le créateur de la vidéo avant de consentir à cette action.

9. Attaque par Remote Code Execution (RCE)

9.1. Principe

Remote Code Execution (RCE) est une vulnérabilité qui permet aux attaquants d'exécuter du code arbitraire sur un serveur distant, leur offrant ainsi un accès non autorisé et la possibilité de compromettre l'intégrité du système. Une vulnérabilité RCE est souvent exploitée lorsqu'un attaquant injecte du code malveillant dans un champ basé sur une requête au sein d'une application web, comme un paramètre d'URL ou un champ de saisie. Lorsque cette requête est transmise au serveur via un intermédiaire, celui-ci exécute les commandes comme si elles provenaient d'un utilisateur légitime, puis renvoie une réponse à l'attaquant. Par conséquent, comprendre et détecter efficacement ces vulnérabilités est essentiel pour les développeurs web, les professionnels de la sécurité et les organisations cherchant à renforcer la protection de leurs actifs numériques contre les cyberattaques. Ainsi, les méthodes d'exploitation des vulnérabilités RCE basées sur le Web constituent une menace majeure, car elles permettent aux attaquants d'exécuter du code arbitraire sur des applications vulnérables. Parmi ces méthodes, on distingue l'exploitation via la méthode $GET et l'exploitation via la méthode $POST, qui reposent sur l'injection de commandes malveillantes dans les requêtes envoyées au serveur, Un scénario d'exploitation RCE est illustré dans la figure suivante. (Biswas, Sajal, Afrin, Bhuiyan, & Hassan, 2018)

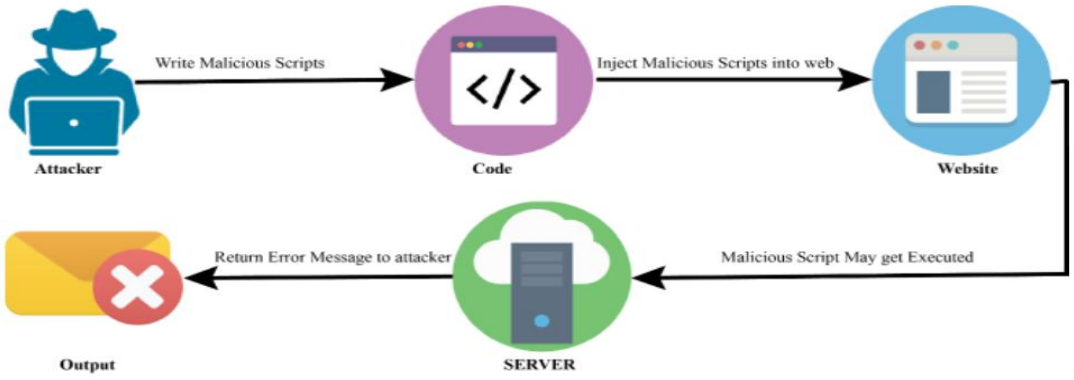

Figure 10: **Processus de RCE**

9.2. Méthodes traditionnelles de protection

Il est souvent difficile de savoir si une tentative d'intrusion vise à exploiter vos

ressources ou à causer des dommages plus graves. Quelle que soit l'intention de l'attaquant, il est essentiel d'adopter des mesures de prévention efficaces. Voici les principales recommandations.

- **Surveillance et alertes précoces**

Même si l'on ne peut pas toujours empêcher l'attaque, des systèmes de détection permettent d'identifier rapidement tout comportement suspect ou l'exécution de code malveillant et d'agir immédiatement.

- **Pare-feu et outils de sécurité**

Déployez un pare-feu d'application web (Web Application Firewall-WAF), complétez-le par des analyses de vulnérabilités et des tests de pénétration afin de repérer et corriger les failles avant qu'elles ne soient exploitées.

- **Plan de réponse aux incidents**

Préparez un protocole clair pour stopper l'attaque, limiter les dégâts et rétablir les services au plus vite.

- **Approche proactive de gestion des vulnérabilités**

La politique la plus décisive consiste à surveiller en continu, mettre à jour régulièrement, tester périodiquement et être prêt à intervenir. Cette démarche globale renforce considérablement la défense contre tout code exécuté à distance. (L'équipe de relations publiques de TuxCare, 2023)

10. Attaque par File inclusion

Les attaques par inclusion de fichiers sont exploitées par un attaquant pour inclure et exécuter un fichier, soit localement, soit à distance. Les deux principaux types d'attaques par inclusion de fichiers sont l'inclusion de fichiers à distance (Remote File Inclusion-RFI) et l'inclusion de fichiers locaux distance (Local File Inclusion-LFI). Nous allons voir les deux types suivants.

10.1. Attaque par Local file inclusion (LFI)

LFI se distingue de RFI par le fait qu'elle exploite des fichiers déjà présents sur le serveur cible. Cette vulnérabilité résulte souvent d'erreurs commises par les développeurs de sites web ou d'applications.

Il est fréquent que les développeurs aient besoin d'inclure des fichiers supplémentaires sur le serveur, tels que des fichiers de configuration, des modules applicatifs ou des fichiers téléchargés par les utilisateurs, comme des images ou des documents texte. Ces fichiers sont généralement stockés dans le répertoire de l'application ou l'un de ses sous-répertoires. Pour y accéder, les développeurs passent souvent les noms des fichiers via des paramètres d'entrée utilisateur, ce qui peut introduire des failles de sécurité exploitables par des attaques LFI. (File Inclusion, 2023)

10.2. Attaque par Remote File Inclusion (RFI)

C'est un type de vulnérabilité qui se produit lorsqu'une application inclut des fichiers externes sans valider ni filtrer correctement les entrées. Cela peut permettre à un attaquant d'injecter du code malveillant ou des URL, entraînant l'exécution de code arbitraire sur le serveur. La RFI est souvent observée dans les applications web et les API écrites dans des langages de programmation tels que PHP, Java Server Pages (JSP) et Active Server Pages (ASP). Une attaque RFI exploite une faille dans une application web en lui faisant inclure un fichier distant malveillant via une requête HTTP spécialement conçue. L'attaquant identifie une application vulnérable, crée une URL pointant vers un fichier malveillant, puis l'envoie en tant qu'entrée utilisateur. Si l'application ne filtre pas correctement cette entrée, elle exécute le code du fichier distant, ce qui peut entraîner la compromission du système, la perte de données et l'accès non autorisé à des informations sensibles. (Senyuz, Eroglu, & Yildiz)

10.3. Méthode traditionnelle de protéction

Afin de prévenir les attaques par inclusion de fichiers (RFI – LFI), il est essentiel de suivre un ensemble de bonnes pratiques et de méthodes de protection éprouvées. Il est crucial de valider et nettoyer soigneusement les entrées utilisateur, en privilégiant l'utilisation de listes blanches pour contrôler les caractères autorisés. L'inclusion de fichiers doit être limitée à un répertoire spécifique et sécurisé, avec une liste blanche de fichiers et de types autorisés. Il faut également restreindre les permissions d'exécution dans les dossiers de téléchargement,

contrôler les types et tailles des fichiers uploadés, et améliorer les configurations du serveur pour éviter l'exécution automatique de fichiers. Le stockage des fichiers sensibles dans une base de données plutôt que sur le serveur constitue une bonne pratique. Enfin, il est recommandé de réaliser régulièrement des tests de sécurité pour détecter toute vulnérabilité liée à l'inclusion de fichiers. (Senyuz, Eroglu, & Yildiz)

11. Attaque par Path Traversal (Directory Traversal)

11.1. Principe

L'attaque Path Traversal, également appelée Directory Traversal, est une attaque visant les applications web du côté serveur. L'attaquant exploite cette vulnérabilité à des fins malveillantes afin de sortir du répertoire racine du serveur en modifiant les paramètres de la requête, lui permettant ainsi de naviguer dans l'arborescence des fichiers et d'accéder à des données sensibles ou des fonctionnalités non autorisées.

Cette attaque est réalisée en trompant le serveur web et en exploitant les paramètres HTTP, le plus souvent via des requêtes GET, ce qui permet à l'attaquant d'accéder à des fichiers situés en dehors des restrictions normalement imposées par l'application. Il peut utiliser ../ pour naviguer entre les répertoires et contourner les restrictions, avec des encodages alternatifs comme %2F pour / en URL, %u2216 en Unicode, ou encore des encodages doubles comme %252e%252e%252f. Certaines failles d'interprétation permettent aussi l'exploitation de codes comme %c0%af, tandis que sous Windows, l'utilisation de \.. au lieu de ../ peut être efficace en raison de la gestion spécifique des chemins. (OWASP Foundation, n.d.)

11.2. Exemple d'attaque Path Traversal

Une faille Path Traversal peut être exploitée pour accéder à des fichiers sensibles sur un serveur web. Cela se fait en modifiant le chemin du fichier demandé dans l'URL, permettant ainsi à l'attaquant de contourner le répertoire de base et d'accéder à des données non autorisées. Voici une comparaison entre les requêtes légitimes et les requêtes malveillantes (voir figure 11):

- **Utilisateur légitime**
  - Un utilisateur classique demande un fichier valide via le lien : http://SiteWeb.com/?file=index.php
  - Le serveur web récupère et affiche correctement index.php depuis son répertoire prévu.

➢ **Attaque Path Traversal**

- L'attaquant modifie l'URL en injectant des séquences ../ pour naviguer dans l'arborescence des fichiers et accéder à des fichiers restreints : http://SiteWeb.com/?file=../../../../etc/passwd
- Si le serveur web n'est pas correctement sécurisé, cette requête sera interprétée et permettra d'accéder au fichier /etc/passwd, qui contient des informations sensibles sur les utilisateurs du système.

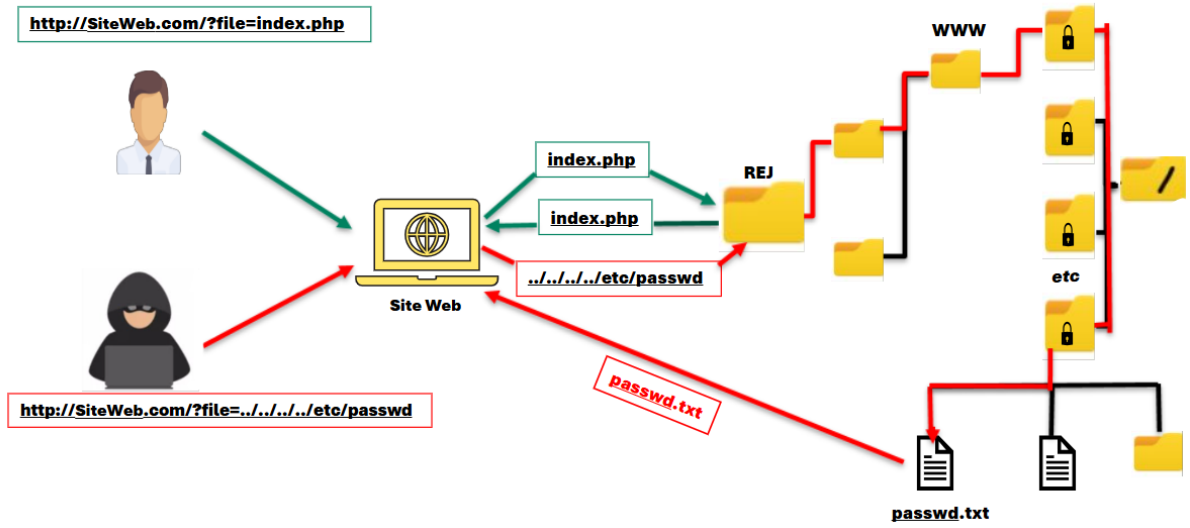

Figure 11 : Exemple d'attaque Path Traversal

11.3. Méthodes traditionnelles de protection

La prévention contre les attaques Path Traversal repose sur un ensemble de bonnes pratiques qui visent à limiter la manipulation des chemins de fichiers par les utilisateurs. Ces méthodes visent à empêcher l'accès non autorisé à des fichiers en dehors du répertoire prévu.

- **Validation stricte des entrées utilisateur**

L'une des protections fondamentales contre les attaques de traversée de répertoires est la validation rigoureuse des données entrées par l'utilisateur. Il convient de vérifier que les paramètres ne contiennent pas de séquences suspectes telles que ../, ..\\ ou leurs équivalents

encodés (%2e%2e%2f, %c0%af, etc.). L'utilisation de listes blanches — qui limitent les valeurs autorisées à un ensemble défini de noms de fichiers ou d'identifiants — permet également de réduire significativement la surface d'attaque. Toutefois, cette méthode peut être contournée si des techniques d'encodage sont mal gérées ou si la validation est insuffisante.

- **Conversion des chemins en format canonique**

Lors de la gestion des chemins d'accès aux fichiers, il est recommandé de convertir les chemins relatifs en chemins absolus (ou canoniques) à l'aide de fonctions comme realpath () en PHP. Une fois cette conversion effectuée, il est essentiel de vérifier que le chemin obtenu reste strictement dans le répertoire autorisé défini par l'application. Cette approche permet d'éviter toute tentative d'évasion du dossier racine, même en présence de séquences malicieuses telles que ../. Toutefois, cette méthode peut avoir un impact sur la performance, notamment si les fichiers sont nombreux ou si les données sont volumineuses, ce qui pourrait ralentir le traitement des requêtes.

12. Attaque par Command Injection

12.1. Principe

L'injection de commandes (Command Injection) est l'une des attaques les plus dangereuses visant les applications interagissant avec le système d'exploitation. Elle permet aux attaquants d'exécuter des commandes non autorisées en exploitant une validation insuffisante des entrées. Ces commandes malveillantes peuvent être transmises via les en-têtes HTTP, les formulaires, les cookies ou les paramètres de requête, et peuvent également provenir de sources externes apparemment fiables mais contrôlées par l'attaquant. Cette attaque repose sur l'exécution directe de commandes au sein de l'interpréteur de commandes du système, permettant ainsi à l'attaquant d'accomplir diverses actions, comme l'envoi d'e-mails ou l'exécution de scripts en Python ou Perl. Il existe plusieurs méthodes d'exécution, notamment le Shell inversé (Reverse Shell), où la machine cible envoie un accès shell à l'attaquant, et le Shell lié (Bind Shell), où le système compromis exécute un shell auquel l'attaquant peut se connecter ultérieurement. Ce type d'attaque représente une menace sérieuse, car il offre à l'attaquant un accès direct au système, pouvant conduire à une prise de contrôle totale de la machine ou au vol de données sensibles. (Bhowmick, 2014)

12.2. Exemple d'attaque Command Injection

Nous avons un site web www.finance.com qui permet aux utilisateurs de compresser et de télécharger leurs données pour une année spécifique. Le problème de sécurité réside dans le fait que le site prend l'entrée de l'utilisateur (l'année) et l'insère directement dans une commande système pour effectuer la compression (zip -r data.zip data-année) sans aucune vérification de sécurité.

L'attaquant exploite cette vulnérabilité en entrant la commande suivante dans le champ de l'année au lieu de saisir une année valide :

2020 ; cat passwd.txt #

Lorsque cette entrée est envoyée, l'application la transmet au système d'exploitation, ce qui entraîne l'exécution de la commande suivante :

zip -r data.zip data-2020; cat passwd.txt #

- La première partie de la commande (zip -r data.zip data-2020) effectue normalement la compression des données.
- La deuxième partie (; cat passwd.txt #) est exécutée après le point-virgule ; et affiche le contenu du fichier passwd.txt.

La figure suivante illustre ce scénario d'exploitation.

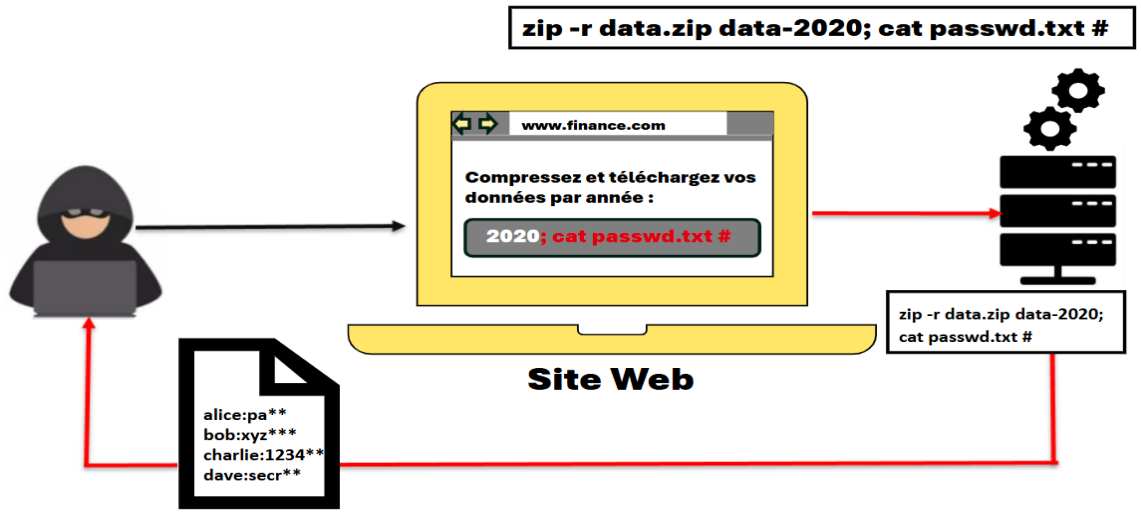

Figure 12 : Exemple d'attaque Command Injection

12.3. Méthode traditionnelle de protection

Une solution traditionnelle pour se prémunir contre les attaques de type Command Injection consiste à éviter l'exécution directe de commandes système via des fonctions comme Runtime.exec() en Java. Il est fortement recommandé d'utiliser les Application Programming Interface (API) natives et sécurisées fournies par le langage, par exemple l'API javax. mail. * pour l'envoi d'e-mails au lieu d'appeler directement la commande système mail. Si aucune API adéquate n'est disponible, le développeur doit alors mettre en place un mécanisme rigoureux de validation des entrées, en appliquant un modèle de sécurité positif — c'est-à-dire en définissant explicitement les caractères autorisés plutôt que d'essayer de bloquer ceux qui sont potentiellement malveillants. Cette approche réduit considérablement la surface d'attaque et renforce la robustesse de l'application face aux tentatives d'injection. (Bhowmick, 2014)

Conclusion

Bien que les technologies de sécurité telles que le chiffrement, l'authentification multi-facteurs et les pare-feux applicatifs soient nombreuses, elles ne garantissent pas une protection totale des applications web, surtout face à l'évolution constante des techniques d'attaque et au manque de sensibilisation à la sécurité chez certains développeurs. Et même si ces solutions traditionnelles ont permis d'améliorer le niveau de sécurité, elles ne suffisent plus à protéger contre les menaces modernes. Cela impose donc un passage vers des approches plus avancées, reposant sur l'intelligence artificielle, grâce à sa capacité à analyser de grandes quantités de données, à détecter les comportements anormaux et à réagir intelligemment face aux attaques complexes. L'intelligence artificielle permet de renforcer efficacement la défense contre ce type d'attaques et contribue à une transformation qualitative des mécanismes de cybersécurité. Ainsi, assurer la sécurité des applications web nécessite une approche préventive intégrée, combinant les meilleures pratiques de développement, des mises à jour régulières, des audits de sécurité fréquents, ainsi que l'intégration de l'intelligence artificielle et de l'apprentissage automatique comme éléments fondamentaux de toute stratégie de cybersécurité à l'ère numérique.

Chapitre 2 : L'apprentissage automatique dans la sécurité des applications web

Introduction

Dans le domaine de la cybersécurité, et plus particulièrement dans le contexte de la détection des attaques web, les datasets jouent un rôle fondamental. En effet, la performance des systèmes de détection d'intrusion basés sur l'apprentissage automatique dépend fortement de la qualité, de la diversité et de la représentativité des données utilisées pour l'entraînement et l'évaluation des modèles. Ce chapitre est consacré à l'analyse des principaux datasets qui ont été utilisés dans les travaux de recherche sur la détection des attaques web.

Nous présentons une description détaillée de plusieurs datasets de référence largement utilisés pour la détection d'intrusions web. Pour chacun d'eux, nous décrivons la nature des données collectées, le type d'attaques couvertes, les caractéristiques disponibles, le format des instances, ainsi que leur pertinence dans le cadre de la cybersécurité.

Enfin, une comparaison détaillée est présentée sous forme de tableau récapitulatif, permettant au lecteur d'identifier les avantages et les limites de chaque dataset. Cette analyse comparative permet d'évaluer la pertinence de chaque dataset dans le contexte des systèmes de détection d'attaques web basés sur l'apprentissage automatique. Ainsi, cette comparaison permettra d'orienter le choix du ou des datasets les plus appropriés pour notre étude, en tenant compte de leur capacité à représenter les attaques web, de la qualité et de la richesse des informations qu'ils offrent, ainsi que de leur compatibilité avec les objectifs fixés dans le cadre de ce travail.

1. Les dataset les plus traités dans les études antérieures

1.1. Le dataset KDD-Cup 99

Le dataset KDD Cup 99 a été préparé en 1999 par Stolfo et al., et il est considéré comme l'un des premiers ensembles de données utilisés dans le domaine de la cybersécurité. Les données ont été capturées à l'aide de tcpdump dans le cadre du programme DARPA IDS en 1998, au sein d'un réseau local simulé (Local Area Network-LAN) par les Lincoln Labs du Massachusetts Institute of Technology (MIT) pendant environ neuf semaines. Ce dataset contient plus de cinq millions d'enregistrements de connexion, classés entre trafic normal et malveillant. Il comprend 41 caractéristiques (features) couvrant différents aspects des connexions réseau, tels que la durée, le protocole utilisé, le nombre de paquets échangés, etc.

Ce dataset comprend 4 898 431 enregistrements de connexions réseau, répartis en deux

grandes catégories : le trafic normal et les attaques malveillantes. Plus précisément, comme le montre le tableau 1 ci-dessous, 80,14 % des connexions sont liées à des attaques, tandis que seulement 19,86 % représentent un trafic normal.

Tableau 1: **Différents types de trafic dans dataset KDD Cup 99**

Type de trafic	**Nombre**	**Pourcentage**
DOS	3883370	79.28 %
Probe	41102	0.84 %
U2R	52	0.0011 %
R2L	1126	0.023 %
Attaques	**3925650**	**80.14 %**
Normal	**972781**	**19.86 %**
Total	**4898431**	**100 %**

On observe à travers le tableau une forte prédominance des attaques de type DOS, qui représentent à elles seules environ 79,28 % de l'ensemble des connexions enregistrées. En revanche, les attaques de type User to Root (U2R) et Remote to Local (R2L) sont très faiblement représentées, ne constituant respectivement que 0,0011 % et 0,023 % des données. Ce déséquilibre important entre les classes pose un véritable défi aux algorithmes d'apprentissage automatique, car il les pousse à favoriser les catégories les plus fréquentes, tout en négligeant celles qui sont rares, bien qu'elles soient souvent plus critiques et complexes à détecter.

Par ailleurs, une analyse critique menée par Tavallaee et al. (2009) a mis en évidence un problème fondamental de redondance dans cette base de données : environ 78,05 % des enregistrements d'entraînement et 75,15 % de ceux du test sont des duplicatas. Cette surreprésentation des exemples fréquents, notamment les attaques de type DOS, engendre un biais d'apprentissage qui nuit à la capacité de généralisation des modèles et à leur performance face à des attaques plus rares et sophistiquées. (Tavallaee,

Bagheri, Ghorbani, & Lu, 2009)

1.2. Le dataset NSL-KDD

Face aux limites identifiées dans Le dataset KDD Cup 99, notamment la redondance et le déséquilibre marqué entre les classes, comme mentionné précédemment, Tavallaee et al. (2009) ont introduit une version améliorée appelée NSL-KDD. Les données proviennent du Canadian Institute for Cybersecurity (CIC), University of New Brunswick (UNB), Canada.

Cette base vise à offrir une évaluation plus fiable des systèmes de détection d'intrusions (IDS), en corrigeant plusieurs biais méthodologiques de KDD'99. Parmi les caractéristiques majeures de NSL-KDD, on note :

- Absence de duplication excessive dans les ensembles d'entraînement et de test, limitant ainsi les biais en faveur des enregistrements fréquents.
- Une distribution plus équilibrée des données selon leur difficulté, permettant d'évaluer plus finement la robustesse des modèles.
- Une taille raisonnable des ensembles, ce qui facilite les expérimentations sans recourir au sous-échantillonnage aléatoire.

- Une vue détaillée de cette répartition est présentée dans le tableau ci-dessous :

Tableau 2: Distribution des requêtes dans NSL-KDD

Type de trafic	Train	Test	Total	Pourcentage
DOS	45 927	7 460	53 387	35.94 %
Probe	11 656	2 421	14 077	9.48 %
U2R	52	67	119	0.08 %
R2L	995	2 885	3 880	2.61 %
Attaques	**58 630**	**12 833**	**71 463**	**48.11 %**
Normal	**67 343**	**9 711**	**77 054**	**51.89 %**
Total	**125 973**	**22 544**	**148 517**	**100 %**

On remarque à travers ce tableau une répartition plus équilibrée entre les connexions normales et malveillantes : les attaques représentent 48,11 % des enregistrements, contre 51,89 % pour le trafic normal. Cette configuration est bien plus réaliste que celle de KDD Cup 99, où

les attaques dominaient largement.

Cependant, certains déséquilibres subsistent. Les attaques U2R (0,08 %) et R2L (2,61 %) restent faiblement représentées, ce qui peut poser des difficultés aux algorithmes pour les apprendre efficacement. De plus, malgré son amélioration, NSL-KDD souffre encore de limitations inhérentes aux datasets synthétiques : il ne reflète pas fidèlement les environnements réseau actuels, n'intègre pas d'attaques modernes (comme les attaques Web ou basées sur des protocoles récents) et reste basé sur des données anciennes.

Ainsi, bien que NSL-KDD constitue une référence précieuse pour le benchmarking, il est essentiel de compléter son usage par des données plus récentes et réelles afin de garantir la validité des systèmes de détection dans des contextes opérationnels modernes. (Tavallaee, Bagheri, Ghorbani, & Lu, 2009)

1.3. Le dataset ECML/PKDD 2007

Le dataset ECML/PKDD 2007 a été développé en 2007 dans le cadre du Discovery Challenge, organisé lors des conférences européennes European Conference on Machine Learning (ECML) et Principles and Practice of Knowledge Discovery in Databases (PKDD). Il a été conçu par les co-organisateurs Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) et LGI2P (École des Mines d'Alès), en collaboration avec la société Bee Ware. Ce dataset est basé sur du trafic web réel et a pour objectif principal de permettre la détection et la classification automatique des attaques dans les requêtes HTTP.

Le Tableau suivant montre la distribution des requêtes dans le dataset ECML/PKDD 2007.

Tableau 3: **Distribution des requêtes dans ECML/PKDD 2007**

Type de trafic	Train	Test	Total	Pourcentage
Cross-Site Scripting	1 813	3 095	4 908	11 %
SQL Injection	2 569	5 064	7 633	17.6 %
LDAP Injection	2 267	4 502	6 769	15.6 %
XPATH Injection	2 267	4 502	6 769	15.6 %

Path Traversal	3 022	5 065	8 087	18.7 %
Command Execution	3 475	6 472	9 947	23 %
SSI Attacks	1 964	3 376	5 340	12.3 %
Attaques	**15 110**	**28 137**	**43 247**	**36.8 %**
Normal	**35 006**	**42 006**	**77 012**	**63.2 %**
Total	**50 116**	**70 143**	**120 259**	**100 %**

Une même requête peut appartenir à plusieurs catégories d'attaque, ce qui fait que la somme des pourcentages dépasse 100 %.

Malgré sa pertinence historique, ce dataset présente plusieurs limitations. Il est ancien, ce qui limite sa représentativité des attaques web modernes. La présence de multi-étiquettes dans certaines requêtes rend la classification plus complexe. De plus, la répartition déséquilibrée entre les classes peut introduire un biais dans les modèles d'apprentissage automatique. Enfin, il ne couvre pas les attaques émergentes ciblant les technologies récentes du web.

1.4. Le dataset CSIC HTTP 2010

CSIC HTTP 2010 est une dataset qui a été généré dans le cadre de la recherche sur la détection des intrusions web. Il a été développé par l'Institut de sécurité de l'information du Conseil national de la recherche d'Espagne. Il contient un total de 61 065 requêtes HTTP, dont 36 000 requêtes normales et 25 065 requêtes anormales. Ainsi, environ 59 % du dataset est constitué de requêtes normales, contre 41 % de requêtes anormales.

Parmi les attaques représentées dans cet ensemble, on retrouve principalement des attaques SQL injection ainsi que des attaques XSS. Le texte complet de chaque requête HTTP a été utilisé dans les expériences réalisées dans le cadre de cette étude. Elle comporte 17 caractéristiques (features), parmi lesquelles celles indiquées dans le Tableau 4. Chacune décrivant des caractéristiques spécifiques des requêtes, permettant une analyse fine pour les tâches de classification et de détection d'anomalies.

Tableau 4: **Analyse des caractéristiques (features) de dataset CSIC HTTP 2010**

Attributs	Valeurs possible	Distribution (%)
Method	GET, POST, PUT	71% GET, 29% POST, 1% PUT
User-Agent	Mozilla	100% Mozilla
Pragma	no-cache	100% no-cache
Cache-Control	no-cache	100% no-cache
Accept	text/xml, null	99% text/xml, 1% null
Accept-Encoding	x-gzip	100% x-gzip
Accept-Charset	UTF-8	100% UTF-8
Language	En	100% en
Host	localhost:80, localhost90	99% localhost:80, 1% localhost90
Cookie	61 065 valeurs uniques	—
Content-Type	null,application/x-www-form-urlencoded	71% null, 29% application/x-www-form-urlencoded
Connection	close, connection: close, other	71% close, 29% connection: close, 28% other
Length	null, content-length, other	71% null, 2% content-length, 28% other
Content	null, B2, other	71% null, 2% B2, 82% other
Class	normal, anomalous	59% normal, 41% anomalous
URL	Chemins d'accès des requêtes web	—

Source : (Gharibeh , Shatha , & Hassan , 2020)

Elle présente plusieurs inconvénients majeurs. Elle est ancienne, ce qui la rend peu fiable pour l'évaluation des systèmes actuels de détection d'intrusion basés sur l'apprentissage automatique. Elle ne contient pas de trafic HTTPS, ce qui limite sa représentativité des environnements web modernes. De plus, elle est conçue uniquement pour la classification binaire (requêtes normales ou malveillantes), ce qui empêche la détection de multiples types d'attaques. La base souffre également d'un déséquilibre entre classes, avec une majorité de requêtes normales, et inclut des échantillons mal encodés. (Sadqi & Chakir, 2024)

1.5. Le dataset UNSW-NB15

Le dataset UNSW-NB15 est l'une des bases de données les plus récentes dans le domaine de la cybersécurité, développée en 2015 par Nour Moustafa et Jill Slay à l'Université de New South Wales (UNSW). Il a été conçu à partir d'un environnement de test synthétique en utilisant l'outil IXIA PerfectStorm, permettant de générer du trafic réseau combinant des comportements normaux réels et des activités malveillantes de manière réaliste et contemporaine. Le Centre Australien pour la Cybersécurité (ACCS) a dirigé ce projet, en collaboration avec des chercheurs internationaux, pour créer le dataset UNSW-NB15. L'outil IXIA PerfectStorm d'ACCS a été utilisé dans le Cyber Range Lab pour créer un mélange de comportements normaux modernes et d'activités d'attaques synthétiques. Le dataset contient 49 caractéristiques extraites à l'aide d'outils tels que Argus et bro-IDS. Le UNSW-NB15 vise à pallier les limites des datasets plus anciens comme KDD-Cup 99 et NSL-KDD, en offrant des caractéristiques avancées et en représentant plus fidèlement des menaces à faible impact. Le dataset contient un total de 2 540 044 enregistrements, avec des sous-catégories d'attaques réparties sur neuf types d'attaques, comme détaillé dans le tableau ci-dessous :

Tableau 5: Distribution des requêtes dans UNSW-NB15

Type de trafic	Description	Nombre	Pourcentage
Fuzzers	Envoi de données aléatoires pour tester les vulnérabilités	24 246	0,95 %
Analysis	Scans de ports, spams, pénétration HTML	2 677	0,11 %
Backdoors	Accès non autorisé via des portes dérobées	2 329	0,09 %
DoS	Tentatives de rendre un service indisponible	16 353	0,64 %
Exploits	Exploitation de vulnérabilités logicielles connues	44 525	1,75 %
Generic	Attaques génériques contre les algorithmes de chiffrement	215 481	8,48 %
Reconnaissance	Collecte d'informations sur le réseau	13 987	0,55 %
Shellcode	Code d'exploitation injecté	1 511	0,06 %
Worms	Vers informatiques autoréplicants	174	0,01 %
Attaques	—	**321 283**	**12,61 %**
Normal	**Transactions normales**	**2 218 761**	**87,39 %**
Total	—	**2 540 044**	**100 %**

Bien que le Dataset UNSW-NB15 soit conçu pour couvrir une variété d'attaques au niveau réseau, il reste limité lorsqu'il s'agit d'analyser les attaques ciblant les applications, telles que XSS ou SQL Injection comme le montre la distribution des requêtes dans le Tableau 5. Cela s'explique par son orientation vers les données réseau sans inclure le contenu des requêtes applicatives. Par conséquent, son efficacité diminue dans les contextes liés à la sécurité des applications web. (Moustafa & Slay, 2015)

1.6. Le dataset HttpParams

HttpParams est une dataset publié en 2015, est l'un des ensembles de données relativement récents utilisés pour l'entraînement des modèles de détection d'attaques sur les applications web. Il se distingue par sa structure simple, composée de seulement quatre colonnes, représentant différents paramètres HTTP, ce qui le rend adapté aux expérimentations initiales et aux modèles légers en termes de complexité. (Shaheed & Kurdy, 2022)
Ce dataset a été construit à partir de plusieurs sources disponibles en libre accès :

- Les valeurs payload des requêtes bénignes ont été extraites du dataset CSIC2010.
- L'outil sql_map a été utilisé pour générer des exemples d'injections SQL.
- L'outil xssya a permis de générer des exemples d'attaques Cross-Site Scripting (XSS).
- Le scanner Vega a été utilisé pour produire des échantillons d'injection de commande et de Path Traversal.
- Le dépôt FuzzDB a également été exploité pour enrichir le dataset avec des attaques supplémentaires de type XSS, Command Injection et Path Traversal. (Morzeux, 2020)

Tableau 6 : Distribution des requêtes dans HttpParams

Type de trafic	Étiquette	Nombre	Pourcentage
Injection SQL	SQLi	10 852	34,93 %
Cross-Site Scripting (XSS)	Xss	532	1,71 %
Injection de commande	Cmdi	89	0,29 %
Traversée de répertoires (Path)	Path-traversal	290	0,93 %
Attaques	**Anom**	**11 763**	**37,85 %**
Normal	**Norm**	**19 304**	**62,15 %**
Total	**—**	**31 067**	**100 %**

Bien que le dataset HttpParams constitue un outil efficace pour l'entraînement des modèles de détection d'attaques sur les applications web, il présente certaines limites. Sa structure simplifiée, composée de seulement quatre colonnes, ne permet pas de refléter la complexité réelle des requêtes observées dans les environnements web modernes. De plus, la répartition des échantillons est déséquilibrée, avec une prédominance marquée des attaques de type SQLi, représentant plus de 90 % des requêtes malveillantes, comme indiqué dans le Tableau 6. Enfin, une utilisation isolée de ce dataset peut limiter la capacité des modèles entraînés à généraliser efficacement dans des contextes réels, en particulier face à des attaques évoluées ou dissimulées.

1.7. Le dataset CIC-IDS 2017

Selon l'auteur du CIC-IDS2017, le dataset est réparti en huit fichiers distincts, contenant à la fois du trafic normal et des attaques, enregistrés sur une période de cinq jours à l'Institut Canadien de Cybersécurité (Ranjit & Borah, 2018), Ce trafic a été généré dans un environnement contrôlé, sur un réseau constitué de quatre machines victimes et de quinze machines attaquantes. Diverses attaques ont été simulées entre le mardi et le vendredi, notamment des attaques de type SQL Injection, XSS, Brute Force, DDoS, Patator, DoS et Heartbleed.

Le Tableau suivant montre la distribution des requêtes dans le dataset CIC-IDS2017 :

Tableau 7: **Distribution des requêtes dans CIC-IDS2017**

Type de trafic	Nombre	Pourcentage
DoS Hulk	231 072	8,46 %
PortScan	158 930	5,82 %
DDoS	41 835	1,53 %
DoS GoldenEye	10 293	0,38 %
FTP-Patator	7 938	0,29 %
SSH-Patator	5 897	0,22 %
DoS slowloris	5 796	0,21 %

DoS Slowhttptest	5 499	0,20 %
Bot	1 966	0,07 %
Web Attack – Brute Force	1 507	0,06 %
Web Attack – XSS	652	0,02 %
Infiltration	36	0,001 %
Web Attack – SQL Injection	21	0,0008 %
Heartbleed	11	0,0004 %
Attaques	**372 163**	**13,63 %**
Normal	**2 359 087**	**86,37 %**
Total	**2 731 250**	**100 %**

Toutefois, le dataset présente plusieurs limites. Il souffre d'un déséquilibre important avec un trafic bénin représentant 86,37 % des données, comme indiqué dans le Tableau 7, ce qui peut fausser l'entraînement des modèles d'intelligence artificielle. De plus, sa grande taille ainsi que le faible nombre d'occurrences pour certaines attaques, telles que Heartbleed, compliquent l'analyse. Par exemple, il ne contient que 0,0008 % de SQL Injection, 0,02 % de XSS et 0,06 % de Brute Force. En revanche, il est particulièrement adapté à l'analyse des attaques DoS/DDoS au niveau de la couche application, grâce à la proportion élevée de ces dernières, représentant 8,46 %. (Sadqi & Chakir, 2024)

1.8. Le dataset CSE-CIC-IDS2018

CSE-CIC-IDS2018 est une dataset les plus importantes et les plus récentes utilisées dans la recherche en cybersécurité, en particulier dans le domaine des systèmes de détection d'intrusion (IDS) basés sur la détection des anomalies. Cette base a été développée par le Canadian Institute for Cybersecurity (CIC) en collaboration avec le Communication Security Establishment (CSE), dans le but de simuler le trafic réseau réel tout en incluant un large éventail d'attaques informatiques. Elle contient 16 233 002 échantillons collectés sur une période de 10 jours de trafic réseau, où le trafic normal représente environ 81,8 %, tandis que les différentes attaques représentent 18,2 % des données.

Le tableau ci-dessous présente la répartition des requêtes dans le dataset CSE-CIC-IDS2018 :

Tableau 8: Distribution des requêtes dans CSE-CIC-IDS2018

Type de trafic	Nombre	Pourcentage
Attaque DDoS – HOIC	737 969	4,54%
Attaque DDoS – LOIC HTTP	618 483	3,81%
Attaque DoS – Hulk	495 107	3,05%
Botnet	306 804	1,89%
Brute Force – FTP	207 782	1,28%
Brute Force – SSH	204 536	1,26%
Infiltration	171 067	1,06%
Attaque DoS – SlowHTTPTest	144 474	0,89%
Attaque DoS – GoldenEye	38 359	0,24%
Attaque DoS – Slowloris	14 610	0,09%
Attaque DDoS – LOIC UDP	9 740	0,06%
Brute Force – Web	3 247	0,02%
Brute Force – XSS	1 785	0,011%
Injection SQL	487	0,003%
Attaques	**2 950 366**	**18,2%**
Normal	**13 282 636**	**81,8%**
Total	**16 233 002**	**100 %**

Malgré la richesse et la diversité du dataset CSE-CIC-IDS2018, elle présente certains défis et limites. L'un des principaux problèmes est le déséquilibre des classes (Class Imbalance), où le trafic normal représente la majorité des données, avec une proportion de 81,8 %, comme le montre la répartition des données dans le Tableau 8, par rapport aux types d'attaques rares, comme l'injection SQL, qui ne représente que 0,003 % des données. De plus, la taille importante des données, avec un total de 16 233 002 échantillons, constitue un défi pour l'analyse en temps réel, nécessitant des modèles plus efficaces pour gérer de grandes quantités de données. Cette base de données a été spécifiquement conçue pour couvrir à la fois les attaques réseau et celles ciblant les applications web. Cependant, elle est principalement adaptée à l'analyse des attaques réseau, comme les attaques DDoS et le scanning de ports, avec un focus moins marqué sur les

attaques spécifiques aux applications web, ce qui la rend plus similaire au dataset CIC-IDS2017. Bien qu'elle puisse être utilisée dans le domaine de la sécurité des applications, elle est davantage orientée vers l'analyse des menaces au niveau réseau. (Elhanashi, 2023)

1.9. Le dataset Hybrid (CSIC_HTTPParams)

Dans le cadre des efforts visant à fournir à la communauté scientifique des ensembles de données diversifiés, riches et plus représentatifs des environnements réels, un dataset hybride a été construit à partir de deux sources largement reconnues dans le domaine de la sécurité des applications web : CSIC HTTP 2010 et HTTPParams 2015.

Ce dataset hybride a été conçu pour combler les lacunes identifiées dans chaque source prise individuellement, telles que la structure simplifiée de HTTPParams ou la couverture limitée des types d'attaques dans CSIC. En associant les deux, on obtient un ensemble de données plus équilibré, généralisable, et représentatif des scénarios d'attaques variés.

Les deux datasets ont été prétraités, nettoyés, convertis en format CSV, puis fusionnés et mélangés de manière aléatoire. Ensuite, une ingénierie des caractéristiques (features engineering) a été appliquée pour extraire quatre caractéristiques numériques à partir du contenu brut des requêtes HTTP (URL, en-têtes, payload, fichiers) :

- Longueur du contenu (payload_len)
- Ratio des caractères alphanumériques (alpha)
- Ratio des caractères spéciaux (non_alpha)
- Poids de l'attaque (attack_weight) – une mesure composite basée sur des sous-métriques (mots-clés malveillants, accès non autorisés, extensions de fichiers dangereuses, etc.)

Le fichier final contient donc 5 colonnes numériques : les quatre caractéristiques précédentes, plus la colonne label qui indique si la requête est normale (0) ou anormale (1).

Le Tableau 9 suivant montre la distribution des requêtes dans le dataset CSIC_HTTPParams. Il est à noter que seules ces informations sont disponibles, ce qui reflète un manque de détails supplémentaires sur la nature des requêtes ou leurs caractéristiques spécifiques. (Shaheed & Kurdy, 2022) :

Tableau 9: Distribution des requêtes dans CSIC_HTTPParams

Type de trafic	Nombre	Pourcentage
Attaque	19215	28,82 %
Normal	47454	71,18 %
Total	66669	100 %

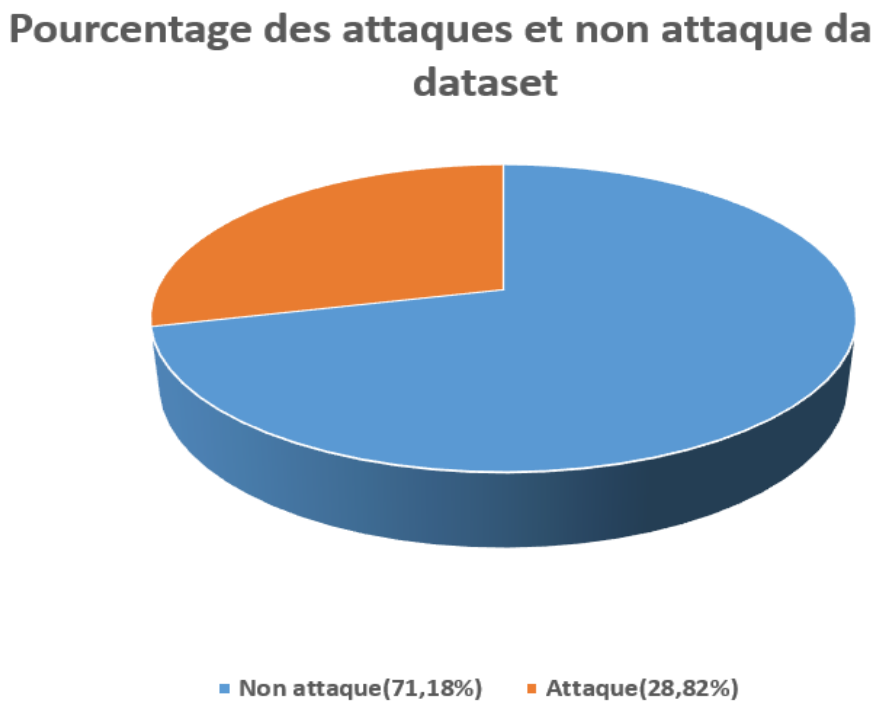

Figure 13: Pourcentage des attaques et non attaques dans le dataset SR-BH 2020 dataset

Le graphique montre un déséquilibre dans le dataset, avec 71,18 % de données normales contre 28,82 % d'attaques. Ce déséquilibre peut compliquer la détection des attaques par les modèles d'apprentissage automatique et nécessite des techniques de rééquilibrage.

1.10. Le dataset SR-BH 2020

Le dataset SR-BH 2020 a été créé par une équipe de recherche conjointe de l'Université

d'Alcalá et de l'UNIR en Espagne. Il s'agit d'un dataset multi-étiquettes (multi-label) visant à classer les attaques web selon la norme CAPEC qui est une base de données maintenue par le MITRE et destinée à répertorier, structurer et décrire les différents types de techniques d'attaque utilisées en cybersécurité. Les données ont été collectées pendant une période de 12 jours en juillet 2020 à partir d'un serveur web fonctionnant sur la plateforme WordPress, en utilisant l'outil ModSecurity dans sa version 2.9.2 et avec le paramètre "mode détection uniquement", permettant d'enregistrer toutes les requêtes, qu'elles soient normales ou malveillantes, sans les intercepter.

Le dataset se compose de 918 176 requêtes web, dont 525 195 requêtes normales (57,85 %) et 392 981 requêtes anormales (42,15 %), ce qui représente une répartition relativement équilibrée, utile pour construire des modèles d'apprentissage automatique solides et réalistes. Les requêtes anormales ont été classées en différentes catégories d'attaques en utilisant le standard CAPEC internationalement reconnu. Certaines requêtes de ce dataset peuvent contenir plusieurs attaques en même temps, ce qui reflète la complexité des attaques modernes.
Le tableau suivant présente la distribution des requêtes dans le dataset SR-BH 2020 :

Tableau 10: **Distribution des requêtes dans SR-BH 2020**

Type de trafic	**Étiquette**	**Nombre**	**Pourcentage**
Manipulation de protocole	272 - Protocol Manipulation	9 153	1,00 %
Injection de code	242 - Code Injection	15 827	1,74 %
OS Command Injection	88 - OS Command Injection	7 482	0,82 %
Path Traversal	126 - Path Traversal	20 992	2,31 %
SQL Injection	66 - SQL Injection	250 311	27,57 %
Dictionary-based Password Attack	16 - Dictionary-based Password Attack	1 847	0,20 %
Scanning for vulnerable software	310 - Scanning for vulnerable software	2 718	0,30 %
Input Data Manipulation	153 - Input Data Manipulation	2 272	0,25 %
HTTP Verb Tampering	274 - HTTP Verb Tampering	5 437	0,60 %
Fake the source of data	194 - Fake the source of data	56 145	6,18 %
HTTP Response Splitting	34 - HTTP Response Splitting	19 738	2,17 %
HTTP Request Smuggling	33 - HTTP Request Smuggling	1 059	0,12 %

Attaques	**—**	392 981	42,15 %
Normal	**000 - Normal**	**525 195**	**57,85 %**
Total	**—**	**918 176**	**100 %**

Bien que le SR-BH 2020 offre une base de données riche et complète couvrant une large gamme d'attaques, y compris celles liées aux réseaux et aux applications web, il existe certaines limitations à prendre en compte. L'une des principales limites est le déséquilibre entre les classes, où certaines attaques, telles que l'injection SQL, dominent considérablement les données, ce qui peut affecter la capacité des modèles à détecter les attaques moins courantes. De plus, la complexité des attaques dans ce dataset, où certaines requêtes peuvent contenir plusieurs attaques simultanément, augmente la difficulté d'une classification précise. En outre, le volume important des données (918 176 requêtes) constitue un défi en termes de ressources informatiques, car l'entraînement des modèles nécessite des systèmes puissants et un temps d'entraînement important. Ces limitations peuvent affecter l'efficacité des modèles et nécessitent des améliorations supplémentaires dans la gestion des données. En ce que suit la description du dataset SR-BH 2020 :

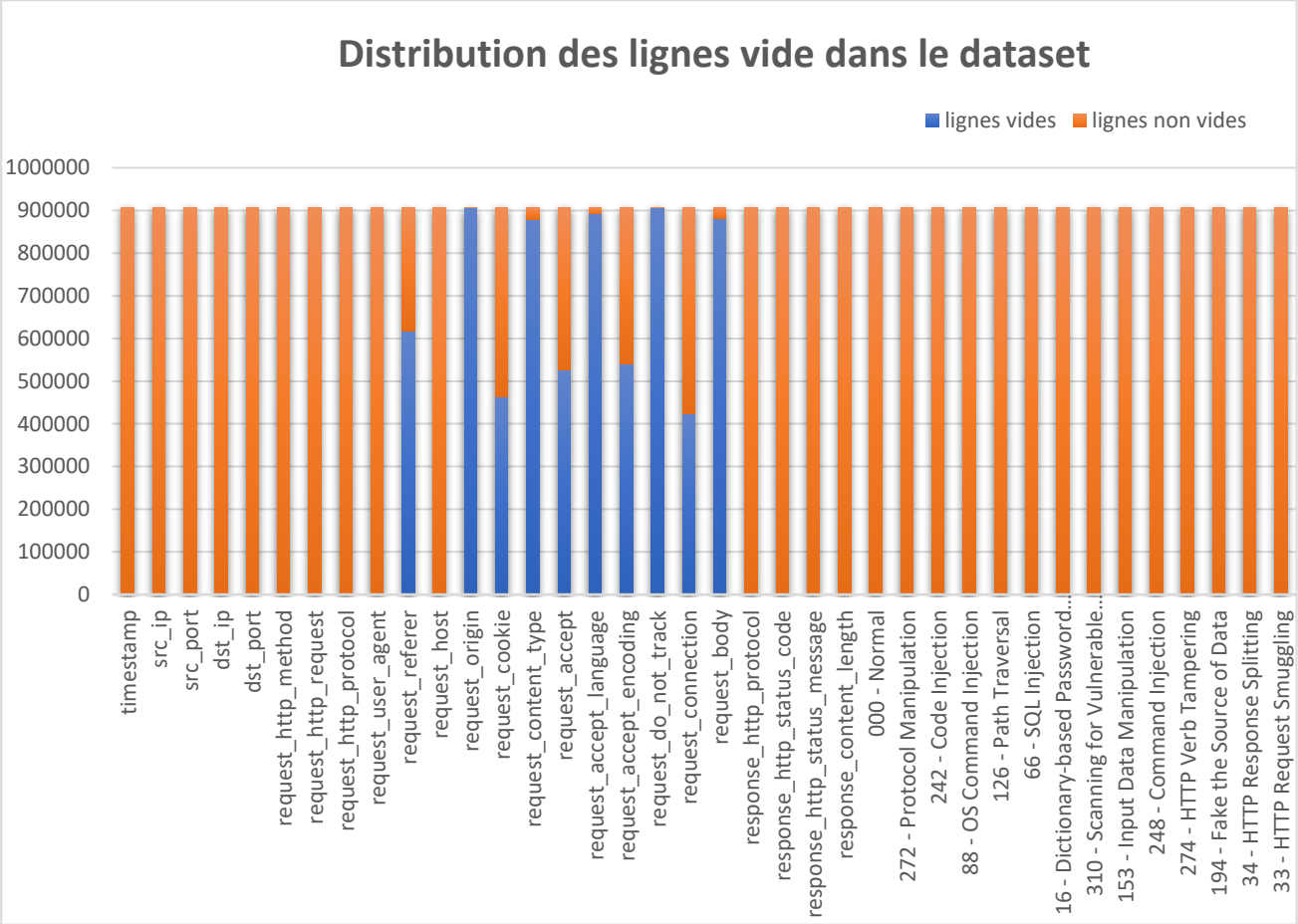

Figure 14 : Distribution des lignes vide dans le dataset

Cet histogramme illustre la distribution des lignes vides (barres bleues) et non vides (barres orange) dans les colonnes d'un dataset, révélant que des colonnes comme timestamp, src_port, dst_port, request_http_request, request_user_agent, et 000 - Normal sont entièrement complètes, tandis que d'autres, telles que request_cookie, request_accept, request_accept_encoding, response_http_protocol, response_http_status_message, et request_connection, présentent une forte proportion de lignes vides, parfois plus de la moitié. Ce déséquilibre met en lumière la nécessité d'un prétraitement (nettoyage, imputation ou suppression) de ces colonnes pour éviter d'impacter négativement l'analyse ou les performances des modèles d'apprentissage automatique.

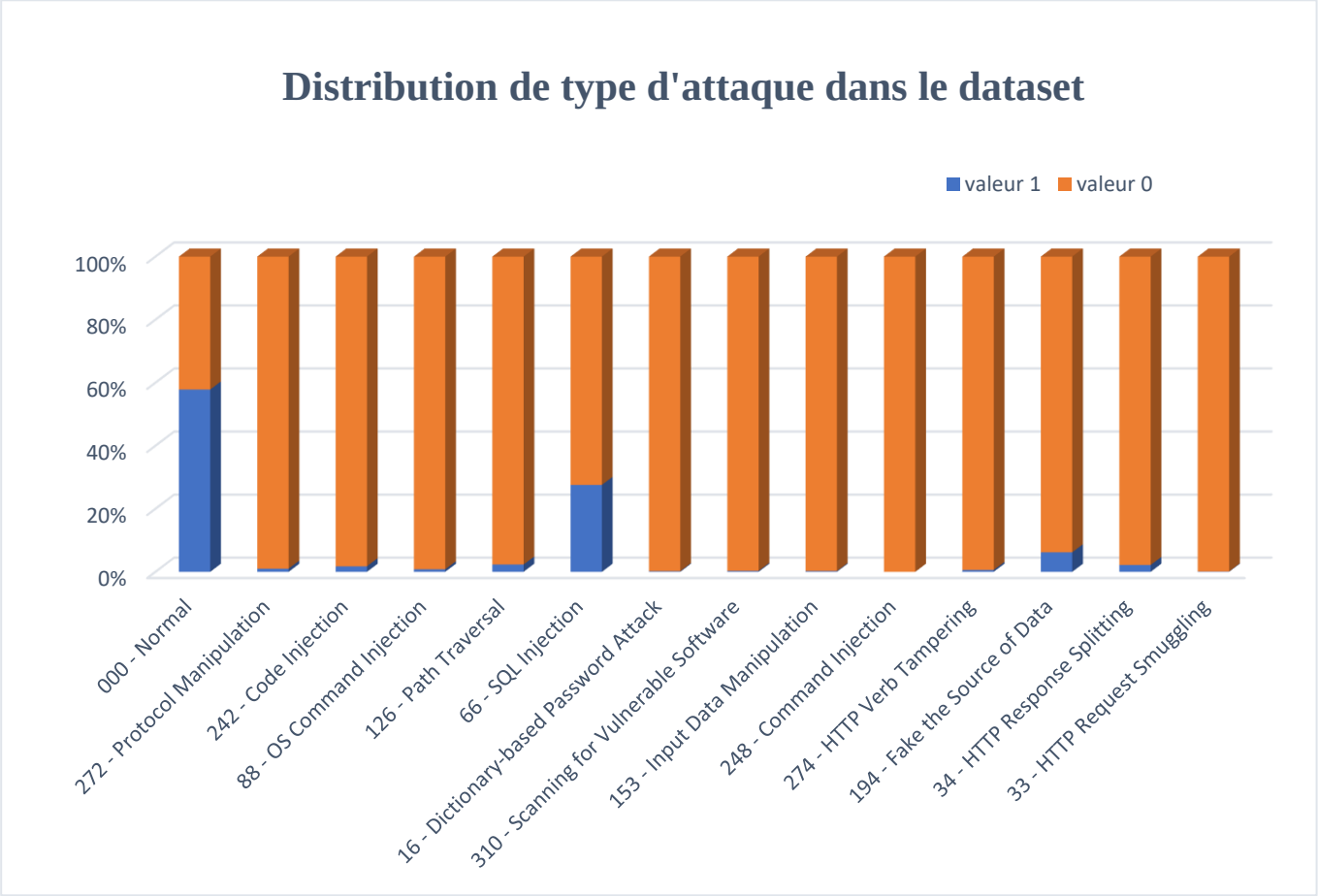

Figure 15:distribution de type d'attaque dans le dataset

Cet histogramme montre une forte prédominance des requêtes normales, suivies des attaques SQL Injection. Les autres types d'attaques sont nettement moins représentés, ce qui indique un déséquilibre important dans la distribution des classes du dataset.

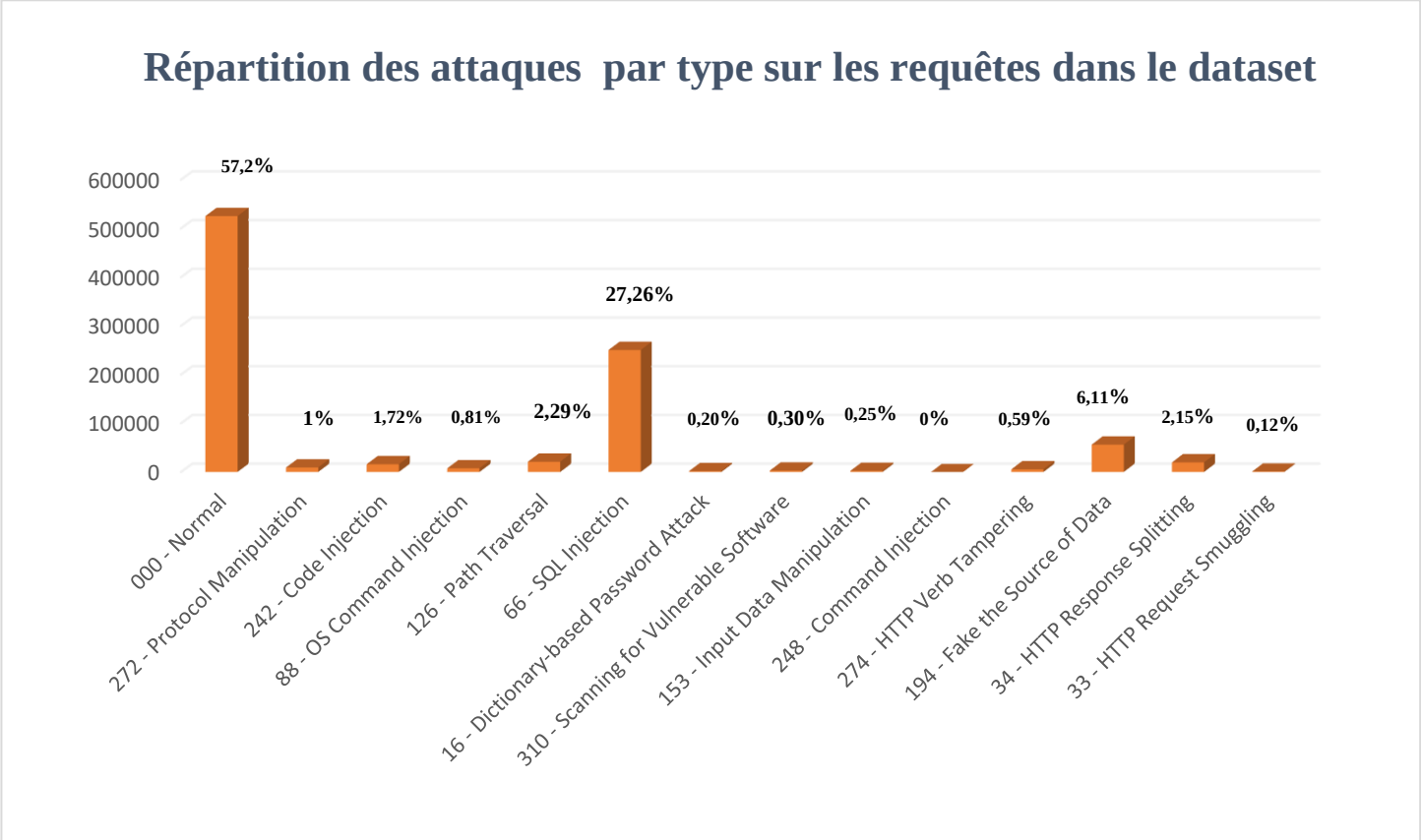

Figure 16 : Répartition des attaques par type sur les requêtes dans le dataset

L'histogramme met en évidence une nette prédominance des requêtes normales (57,2 %) dans le dataset. Les attaques les plus fréquentes sont les SQL Injection (27,26 %), tandis que d'autres, comme Path Traversal (2,29 %), HTTP Response Splitting (2,15 %) et Code Injection (1,72 %), apparaissent de façon marginale. Plusieurs types d'attaques, tels que Command Injection ou HTTP Request Smuggling, sont très peu représentés (moins de 1 %). Cette distribution inégale reflète une forte hétérogénéité dans les types d'attaques présents, ce qui peut entraîner un déséquilibre lors de l'entraînement de modèles de détection d'intrusion.

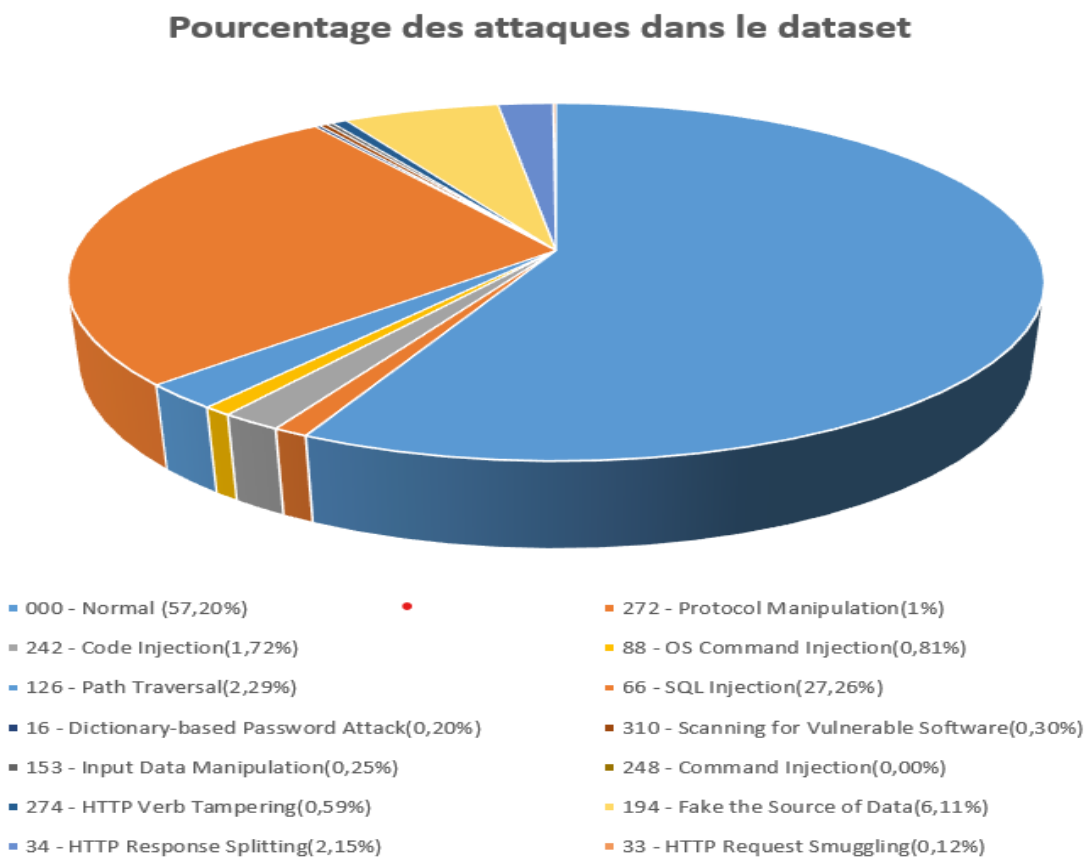

Figure 17:**Pourcentage des attaques dans le dataset**

Le diagramme montre que la majorité des données (57,20 %) sont normales, suivies principalement par les attaques de type SQL Injection (27,26 %). Les autres attaques, comme "Fake the Source of Data", "Path Traversal" ou "Code Injection", sont présentes en plus faibles proportions. Certaines, comme "Command Injection", sont quasi absentes. Ce constat souligne l'importance de prioriser la détection des attaques fréquentes tout en restant attentif aux menaces plus rares.

2. Comparaison entre les datasets

Dans le cadre de notre étude sur la sécurité des applications web à travers des techniques d'apprentissage automatique, nous allons comparer plusieurs datasets représentatifs des attaques réelles ou simulées survenant dans des environnements web. L'accent sera mis sur des datasets récents tels que SR-BH 2020 et Hybrid CSIC_HTTP Params, tout en analysant d'autres datasets moins adaptés, comme NSL-KDD et HTTP Params 2015. La sélection de ces datasets

repose sur plusieurs critères clés, notamment la récence des données, essentielle pour refléter l'évolution des techniques d'attaque, et la diversité des types d'attaques, telles que XSS, SQL Injection et CSRF. Un autre facteur important est la qualité des labels associés aux échantillons, car des labels précis sont indispensables pour garantir une classification correcte des attaques.

Un critère complémentaire va également guider notre choix : l'équilibre entre les catégories d'attaques et non-attaques. Un bon équilibre est essentiel pour améliorer l'efficacité des modèles d'apprentissage automatique et réduire les biais dans les prédictions. Cela nous amène à un constat intéressant concernant les datasets KDD Cup 99 et NSL-KDD, appartenant à la famille KDD. En effet, KDD Cup 99 souffre d'une forte redondance et d'un déséquilibre notable entre les types d'attaques, ce qui affecte négativement la performance des modèles de détection d'intrusions. En réponse à cela, le dataset NSL-KDD a été proposé pour corriger ces faiblesses, notamment en réduisant les duplications et en assurant une meilleure répartition des données. Toutefois, il demeure limité par la faible représentation des attaques rares et par l'ancienneté de ses données, ce qui reste problématique face à l'évolution rapide des menaces actuelles.

Cela nous conduit à la remise en question de l'utilité des datasets historiques, comme ceux de l'Université de Californie à Irvine, qui déconseille l'utilisation des datasets KDD, les qualifiant d'inadaptés et obsolètes pour la détection des menaces modernes. En parallèle, nous avons examiné d'autres datasets bien connus dans le domaine de la cybersécurité, tels que ECML/PKDD 2007, CSIC HTTP 2010 et HttpParams. Bien que ces datasets couvrent une large variété d'attaques, ils présentent plusieurs limitations. L'ancienneté des données, l'absence de couverture des attaques récentes et la non-prise en charge de protocoles modernes comme HTTPS en font des modèles moins efficaces pour détecter les menaces actuelles.

Dans cette même logique, certains datasets récents, tels que UNSW-NB15, CIC-IDS2017 et CSE-CIC-IDS2018, sont largement utilisés pour évaluer les systèmes de détection d'intrusions. Ces datasets couvrent une large variété d'attaques réseau, y compris des attaques de type DoS, DDoS, ainsi que des attaques ciblant spécifiquement les applications web, comme XSS et SQL Injection. Cependant, ces datasets sont principalement orientés vers l'analyse des attaques au niveau du réseau, avec une représentation plus faible des attaques appliquées, ce qui les rend moins adaptés pour la détection d'attaques web.

Dans un autre côté, les datasets plus récents, comme le Hybrid Dataset (CSIC_HTTPParams) et le SR-BH 2020, apportent des solutions plus robustes pour la détection des attaques sur les applications web. Le Hybrid Dataset se distingue par un équilibre et une

complémentarité des données, combinant des informations provenant de plusieurs sources, ce qui améliore la qualité des modèles de détection. De son côté, le SR-BH 2020 propose un dataset multi-étiquettes, couvrant une gamme étendue d'attaques, ce qui permet une meilleure capacité à détecter des menaces complexes. Toutefois, bien qu'ils présentent des avantages notables, ces datasets comportent aussi des limitations, notamment le déséquilibre des classes dans certains cas, ce qui doit être pris en compte pour optimiser leur utilisation.

Ainsi, en analysant ces différents datasets, il apparaît clairement que le choix d'un dataset adapté à la détection des attaques web dépend d'un compromis entre la diversité des attaques, la qualité des données et l'équilibre entre les classes, tout en tenant compte des spécificités de chaque système de détection utilisé.

Après avoir étudié tous ces datasets, certains points de comparaison ont été résumés dans le tableau ci-dessous :

Tableau 11 : **Comparaison entre les datasets.**

Data set	Année	Type de trafic	Features	Type de classification	Séparé Train/Test	Domaine d'application	Distribution des attaques	Pourcentage Attaques	Pourcentage Normal	Total des requêtes
SR-BH 2020	2020	13	38	Multi-label (CAPEC)	Non	Web	Plus équilibrée	42,15 %	57,85 %	918 176
Hybrid CSIC_HTTP Params	2022	2	5	Binaire	Non	Web	Plus équilibrée	28,82 %	71,18 %	66669
CSE-CIC-IDS2018	2018	15	83	Multi classe	Non	Réseau Web	Déséquilibrée	18,2%	81,8%	16227068
CIC-IDS2017	2017	15	79	Multi classe	Non	Réseau Web	Déséquilibrée	13,63 %	86,37 %	2830743
Http Params	2015	5	4	Multi classe	Non	Web	Déséquilibrée	37,85 %	62,15 %	31 067
UNSW-NB15	2015	10	49	Multi classe	Non	Réseau	Déséquilibrée	12,61 %	87,39 %	2540044
CSIC HTTP 2010	2010	2	17	Binaire	Non	Web	Plus équilibrée	41 %	59 %	61 065
ECML/PKDD 2007	2007	8	43	Multi-label	Oui	Web	Déséquilibrée	36.8 %	63.2 %	120 259
NSL-KDD	2009	5	41	Multi classe	Oui	Réseau	Plus équilibrée	48.11 %	51.89 %	148 517
KDD-Cup 99	1999	5	41	Multi classe	Non	Réseau	Déséquilibrée	80.14 %	19.86 %	4898431

3. Choix des datasets pour application de modèles

L'analyse comparative a mis en évidence plusieurs critères déterminants pour le choix d'un dataset pertinent : la récence des données, la diversité des types d'attaques (telles que XSS, SQL Injection, CSRF), la qualité des labels, et l'équilibre entre le trafic normal et malveillant. Ces critères influencent directement les performances des modèles d'apprentissage automatique, en termes de précision, de généralisation et de robustesse.

Sur la base de ces critères, notre choix s'est porté sur deux le dataset : SR-BH 2020 et Hybrid CSIC_HTTP Params. Ces ensembles offrent une couverture riche et représentative des attaques modernes, une bonne qualité des caractéristiques, ainsi qu'un format adapté à l'entraînement et à l'évaluation des modèles de détection d'intrusion. Ce choix constitue ainsi une base solide pour les expérimentations à venir, et contribuera à la construction de systèmes plus efficaces pour sécuriser les applications web.

Conclusion

Ce chapitre a permis d'examiner en détail plusieurs le dataset utilisés dans le domaine de la cybersécurité, en particulier pour la détection des intrusions sur les réseaux et les applications web. Des ensembles classiques comme KDD-Cup 99 et NSL-KDD ont été présentés, tout comme des datasets plus récents et mieux adaptés aux menaces actuelles, tels que UNSW-NB15, CSIC HTTP 2010, HttpParams, CIC-IDS2017, CSE-CIC-IDS2018, ainsi qu'un dataset hybride issu de la fusion de CSIC HTTP 2010 et HttpParams. Nous avons également examiné SR-BH 2020, un ensemble plus récent et spécifiquement orienté vers les attaques sur les applications web. Notre choix a finalement reposé sur les datasets SR-BH 2020 et Hybrid CSIC_HTTP Params, dans le chapitre suivant, nous allons voir le prétraitement de ces datasets, le choix et l'application des modèles d'apprentissage automatique.

Chapitre 3 : La conception d’un système De détection basé sur l’apprentissage automatique

Introduction

Ce chapitre couvre le volet pratique de notre recherche sur les systèmes de détection des attaques web utilisant des techniques d'apprentissage automatique. Nous commencerons par exploiter directement la base de données hybride CSIC_HTTP Params, déjà prétraitée, à laquelle nous appliquerons toutefois certaines modifications, notamment un rééquilibrage des classes. Nous commençons ainsi par cette phase préalable avant de détailler les différentes étapes mises en œuvre pour l'entraînement et l'évaluation des modèles, jusqu'à l'analyse finale de leurs performances. Ensuite, nous exposerons les étapes de prétraitement appliquées au dataset SR-BH 2020, avant de présenter les algorithmes utilisés et, enfin, de comparer les performances obtenues par chaque modèle.

1. Outils et environnement de développement

Dans le cadre de notre travail sur l'apprentissage automatique pour la sécurité des applications Web, nous avons utilisé des outils pour le développement et l'expérimentation. Les langages et plateformes utilisés incluent Python, Jupyter Notebook et Google Colab, qui ont facilité l'analyse des données et l'entraînement des modèles d'apprentissage automatique. Les tests ont été effectués sur deux ordinateurs portables, chacun doté de 16 Go de RAM. Le premier était équipé d'un processeur Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz 2.11 GHz et fonctionnait sous Windows 10, tandis que le second utilisait un Intel(R) Core(TM) i5-1235 U @ 1,30 GHz avec Windows 11 Professionnel.

2. Prétraitement des données

Le prétraitement des datasets constitue une phase cruciale qui impacte directement la qualité et la performance des modèles. Des données mal nettoyées ou incompatibles peuvent engendrer des erreurs et biaiser l'apprentissage. C'est pourquoi nous avons porté une attention particulière à l'exploration, à la vérification et au nettoyage des datasets, afin d'assurer un entraînement optimal.

Ce prétraitement comprend plusieurs activités principales, qui sont appliquées de manière flexible selon la nature des données et les exigences du modèle, notamment :

2.1. Nettoyage des données (Data cleaning)

Cette étape vise à améliorer la qualité des données en supprimant les doublons, en

traitant les valeurs manquantes, et en corrigeant les erreurs de saisie ou incohérences textuelles. Les techniques employées varient selon la structure de chaque dataset, tout en garantissant la cohérence et la fiabilité des informations conservées.

2.2. Transformation des données (Data transformation)

La transformation des données consiste à modifier leur format ou leur structure pour les rendre plus exploitables par les algorithmes de machine learning. Elle inclut notamment le changement de type de variables, l'encodage des données catégorielles, ainsi que la génération de nouvelles caractéristiques à partir des données existantes. Cette étape joue un rôle crucial dans l'amélioration de la qualité de l'apprentissage et des performances du modèle.

2.3. Sélection des caractéristiques (Feature selection)

Cette phase permet d'identifier et de conserver uniquement les variables les plus pertinentes pour la tâche à accomplir. Elle vise à réduire la dimensionnalité des données, améliorer l'efficacité du modèle, limiter les risques de surapprentissage (overfitting) et accélérer le processus d'entraînement. Elle contribue également à rendre le modèle plus interprétable.

2.4. Gestion des données déséquilibrées

Un déséquilibre entre les classes peut biaiser l'apprentissage en favorisant la classe majoritaire. Il est donc essentiel de traiter ce déséquilibre avant l'entraînement afin de garantir une détection équitable des différentes classes. Des techniques de rééchantillonnage, telles que le suréchantillonnage de la classe minoritaire ou le sous-échantillonnage de la classe majoritaire, peuvent être utilisées selon le cas.

2.5. Division du dataset (Data splitting)

Il s'agit de séparer les données en ensembles d'entraînement et de test, de manière à évaluer de façon fiable la capacité de généralisation du modèle. Cette séparation garantit la cohérence des expérimentations et permet de valider les résultats obtenus.

Dans notre étude, nous avons utilisé deux datasets différents. Chaque dataset a été traité spécifiquement en fonction de ses caractéristiques et de sa structure, avec une application adaptée des différentes étapes de prétraitement décrites ci-dessus.

3. Prétraitement des données de dataset (CSIC_HTTPParams)

Au premier abord, ce dataset semblait prêt à l'emploi et bien structuré, ce qui laissait penser que son traitement serait simple et direct. Cependant, après une analyse plus approfondie, nous avons constaté qu'il nécessitait quelques ajustements et nettoyages afin d'assurer la fiabilité des résultats et la précision des modèles.

Ainsi, nous avons suivi une série d'étapes méthodiques pour traiter correctement ces données. Nous présentons ci-dessous les différentes phases du processus que nous avons appliquées pour garantir la qualité et l'efficacité du prétraitement.

3.1. Nettoyage des données (Data cleaning)

Séparation des datasets en ensembles d'entraînement et de test, assurant la cohérence de l'expérience et la fiabilité des résultats.

Dans notre étude, nous avons utilisé deux datasets différents. Chaque dataset a été traité spécifiquement en fonction de ses caractéristiques et de sa structure, avec une application adaptée des différentes étapes de prétraitement décrites ci-dessus.

3.1.1. Vérification des types de données

Nous nous sommes assurés que chaque colonne contenait le type de données approprié, tel que int64 pour la colonne payload_len et float64 pour les colonnes alpha, non_alpha et attack_feature. Cela garantit une utilisation sans erreur pendant la modélisation.

3.1.2. Traitement des valeurs manquantes

Après vérification avec la méthode isnull().sum(), aucune valeur manquante n'a été détectée. Il n'a donc pas été nécessaire d'intervenir à ce niveau.

3.1.3. Analyse des incohérences

Nous avons examiné les valeurs aberrantes ou incohérentes, notamment dans les colonnes alpha et non_alpha où certaines lignes affichaient la valeur 0.0. Un total de 20 032 lignes présentait cette caractéristique. Après analyse, il a été constaté qu'il s'agissait de requêtes HTTP valides, comme celles avec des payloads vides. Ces lignes ont donc été conservées.

3.1.4. Suppression des doublons

L'analyse a révélé 60 289 enregistrements dupliqués. Après suppression, le nombre de lignes est passé de 66 669 à 6 380. Cette étape est cruciale pour éviter les biais lors de l'apprentissage des modèles.

3.2. Sélection des caractéristiques (Feature selection)

Le dataset contient cinq colonnes principales : payload_len, alpha, non_alpha, attack_feature et label. Nous avons conservé les quatre premières comme variables explicatives. Ces caractéristiques ont été soigneusement définies par les auteurs du dataset. Aucune suppression ou modification supplémentaire n'a été nécessaire. La colonne label a été utilisée comme variable cible pour l'entraînement des modèles.

3.3. Division du dataset (Data splitting)

Après nettoyage et sélection des caractéristiques, nous avons divisé le dataset afin d'assurer la cohérence expérimentale et la fiabilité des résultats.

Les données ont été réparties comme suit :

- 80 % pour la partie d'apprentissage (training set)
- 20 % pour la partie de test (test set)

Cette répartition a été effectuée aléatoirement en maintenant la proportion d'origine des classes grâce à la technique de l'échantillonnage stratifié (stratified sampling). Ainsi, chaque sous-ensemble conserve une diversité représentative des types de requêtes, qu'elles soient normales ou malveillantes.

3.4. Gestion des données déséquilibrées

Ce problème est apparu clairement lors de l'analyse du dataset CSIC_HTTPParams, comme illustré dans la figure suivante (distribution des classes avant l'application de SMOTE).

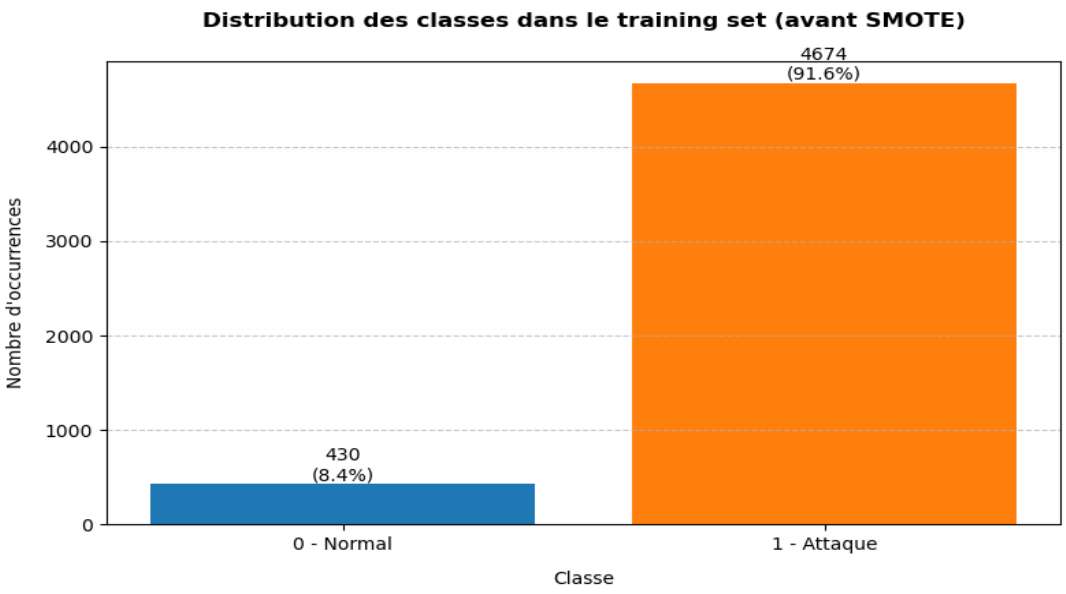

Figure 18 : Distribution des classes dans le training set CSIC_HTTPParams(avant SMOTE)

Pour y remédier, nous avons appliqué la technique Synthetic Minority Over-sampling Technique (SMOTE). Elle génère des exemples synthétiques pour la classe minoritaire à partir des exemples existants. Cette méthode a permis d'équilibrer le dataset d'apprentissage, ce qui améliore la capacité du modèle à détecter les classes sous-représentées.

La figure 19 montre la distribution des classes après l'application de SMOTE, mettant en évidence un équilibre significatif bénéfique pour la robustesse du modèle.

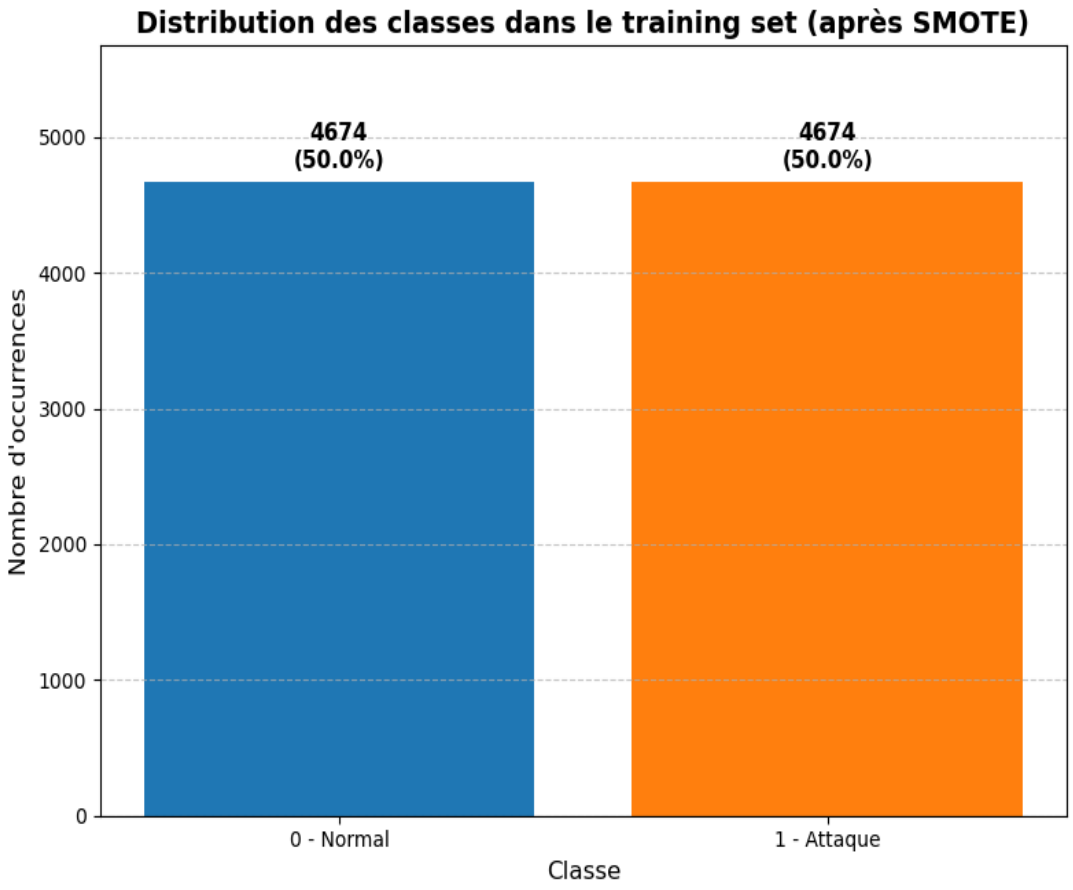

Figure 19 : Distribution des classes dans le training set CSIC_HTTPParams(après SMOTE)

3.5. Transformation des données (Data transformation)

Le dataset utilisé dans cette étude est principalement composé de valeurs numériques et de proportions, ce qui le rend directement exploitable par les modèles de machine learning. Étant donné la nature de ses variables, aucune étape d'encodage n'a été nécessaire, car ces données peuvent être considérées comme déjà encodées ou naturellement compatibles avec les traitements algorithmiques.

4. Prétraitement des données de dataset SR-BH 2020

D'après nos analyses précédentes, le dataset étudié relève initialement d'un problème de classification multi-label, c'est-à-dire que chaque observation pouvait être associée à plusieurs types d'attaques simultanément. Cependant, après avoir effectué une analyse statistique approfondie, nous avons constaté que seulement environ 1 % des lignes du dataset

comportent plus d'une étiquette active (c'est-à-dire plusieurs types d'attaques pour une même instance). Comme le montre la figure 20, cette proportion est très faible par rapport à la taille totale du dataset. **Nous avons donc proposé de supprimer ces cas particuliers afin de simplifier le traitement**. **Une autre raison de choisir le passage à la classification multiclasse est que la classification multilabel est plus complexe et peut réduire les scores globaux**. Cette proposition nous a permis de convertir le dataset d'un format multi-label à un format multi-classe, dans lequel chaque instance est associée à une seule et unique classe. Ce changement facilite grandement l'application des algorithmes de classification classiques et permet une évaluation plus directe et interprétable des performances des modèles prédictifs.

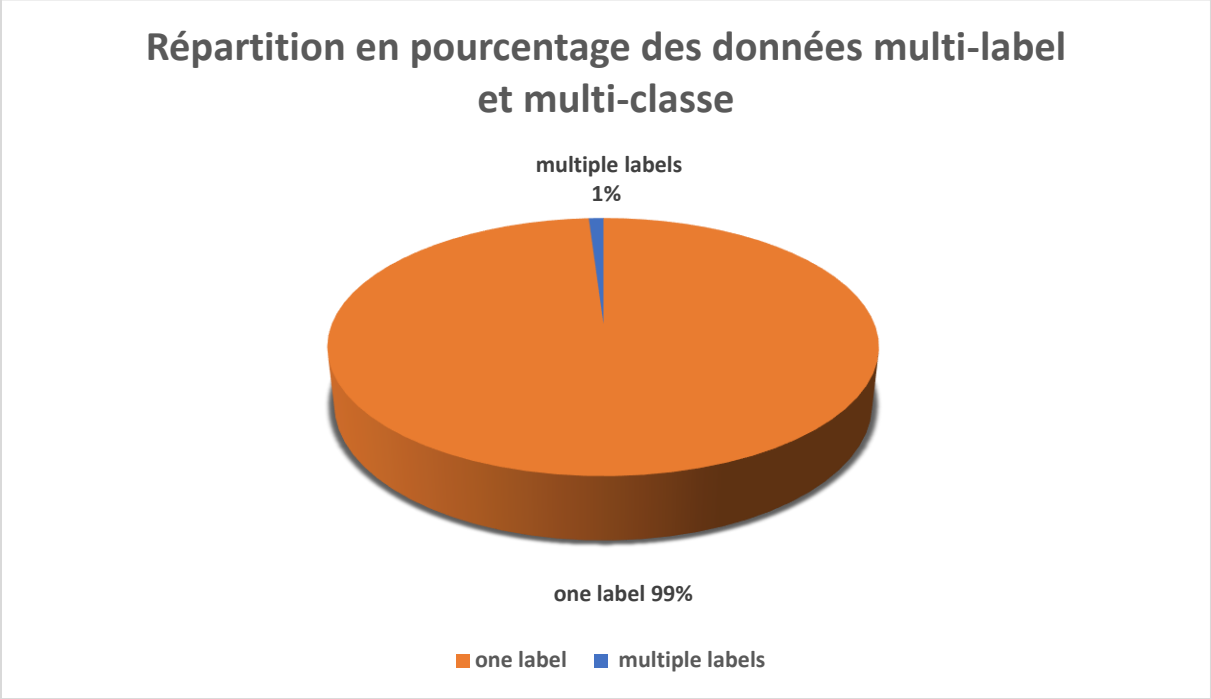

Figure 20 : Répartition en pourcentage des données multi-label et multi-classe

Grâce à cette simplification, le traitement du dataset devient plus clair et mieux adapté aux méthodes classiques de classification. Nous présentons à présent les étapes clés mises en œuvre pour le prétraitement et l'analyse de ce dataset.

4.1. Nettoyage des données (Data cleaning)

Pour le dataset SR-BH 2020, nous avons appliqué un processus de nettoyage rigoureux afin de garantir la qualité et la fiabilité des données pour les étapes ultérieures de l'analyse. Ce

processus comprenait :

4.1.1. La suppression de doublons

L'étape de nettoyage des données a débuté par la suppression des doublons, qui ont été au nombre de 2 860, afin d'éviter qu'ils ne faussent les résultats des modèles d'apprentissage automatique. Nous avons supprimé toutes les lignes en double en conservant uniquement la première occurrence de chaque enregistrement, puis vérifié qu'aucun doublon ne subsistait. Ensuite, nous avons rempli les valeurs manquantes de la colonne response_content_length par zéro, puis converti cette colonne en entier. Après ces opérations, l'ensemble de données comptait 877 133 lignes réparties sur 11 colonnes, sans doublons ni valeurs manquantes dans les colonnes clés.

4.1.2. Filtrage de la variable cible

Dans cette étape, le filtrage de la variable cible consiste à supprimer certaines classes indésirables ou non pertinentes de la variable cible, afin de se concentrer sur les catégories les plus significatives pour l'analyse ou l'entraînement du modèle. Plus précisément, les classes suivantes ont été supprimées du dataset :

'protocol manipulation', 'http verb tampering', 'os command injection', 'scanning for vulnerable software', 'input data manipulation' et 'dictionary-based password attack'. Ces classes ont été supprimées car elles sont faiblement représentées dans le dataset, ce qui risque d'introduire un déséquilibre important et de nuire à la performance du modèle. Leur suppression permet ainsi de réduire le bruit, d'équilibrer la distribution des classes et d'améliorer la qualité de l'apprentissage.

4.2. Sélection des caractéristique (Feature selection)

Dans cette étape, nous procédons à une sélection des caractéristiques les plus pertinentes afin d'optimiser l'apprentissage du modèle de classification. Pour ce faire, nous utilisons la méthode SelectKBest associée au test statistique f_classif, basé sur l'analyse de la variance (ANOVA). Cette approche évalue la capacité de chaque variable à distinguer efficacement les différentes classes de la variable cible. En retenant les 8 caractéristiques les mieux classées, nous réduisons la dimensionnalité du dataset tout en préservant les informations les plus significatives, car nous avons choisi de conserver un plus grand nombre de caractéristiques afin

d'étudier au mieux l'ensemble des variables importantes.

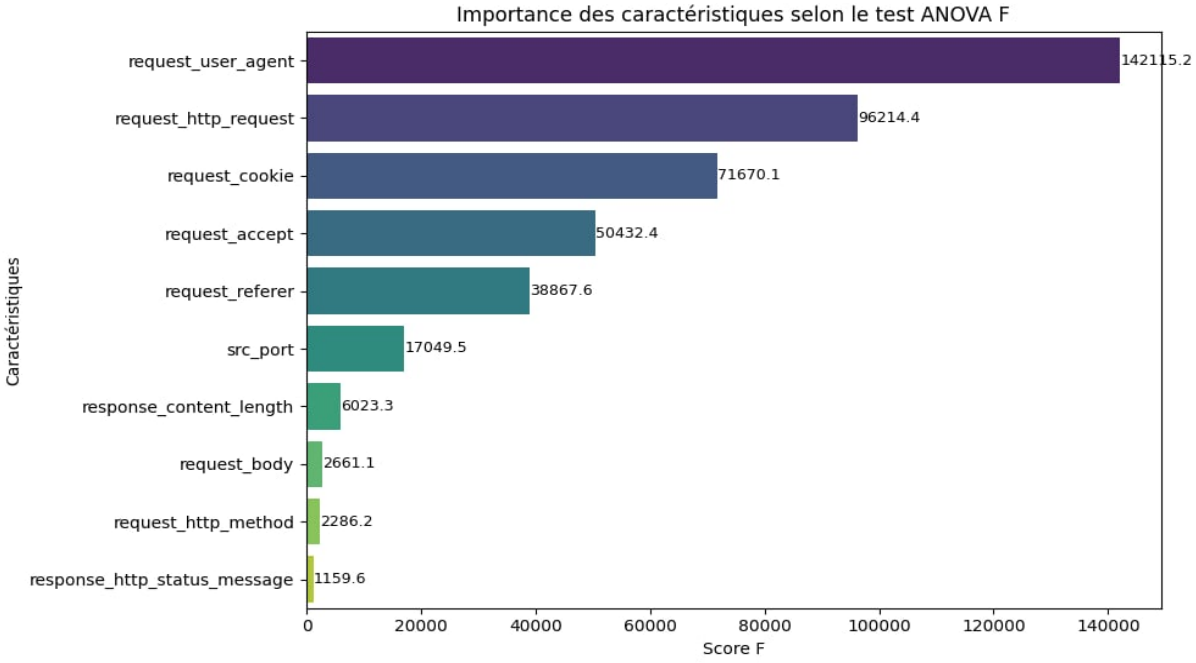

Figure 21 : L'importance des caractéristiques selon le test ANOVA

4.3. Transformation des données (Data transformation)

Cette étape prépare les données catégorielles pour l'analyse en les convertissant en valeurs numériques. Les variables textuelles comme request_http_method et request_user_agent sont encodées numériquement à l'aide du CountEncoder, qui utilise leur fréquence d'apparition. Quant à la variable cible (class), elle est transformée via le LabelEncoder, qui convertit les labels ("normal", "sql injection", etc.) en valeurs numériques (0, 1...), facilitant ainsi leur utilisation par les algorithmes de classification. Cette transformation préserve la signification des données tout en les rendant exploitables par les modèles.

4.4. Division du dataset (Data splitting)

Dans cette étape, les données sont divisées en ensembles d'entraînement (80 %) et de test (20 %) selon la même méthode que celle appliquée sur le premier dataset. Cette procédure uniforme a été mise en place afin d'assurer la cohérence des résultats. Elle permet également de comparer équitablement la performance des modèles sur différentes bases, tout en garantissant une séparation claire entre les données d'apprentissage et celles dédiées à l'évaluation.

4.5. Gestion des données déséquilibrées

Comme il est mentionné précédemment, dans cette étape de la gestion des données déséquilibrées, un problème majeur de déséquilibre des classes a été identifié, comme le montre la figure 22. On remarque que la classe "normal" représente 59,6 % des données, tandis que certaines classes d'attaques, comme "code injection" ou "path traversal", sont fortement sous-représentées. Ce déséquilibre peut fortement impacter les performances des algorithmes d'apprentissage. C'est pourquoi nous avons appliqué la technique SMOTE afin de générer artificiellement de nouvelles instances pour les classes minoritaires et ainsi améliorer l'équilibre du dataset**.**

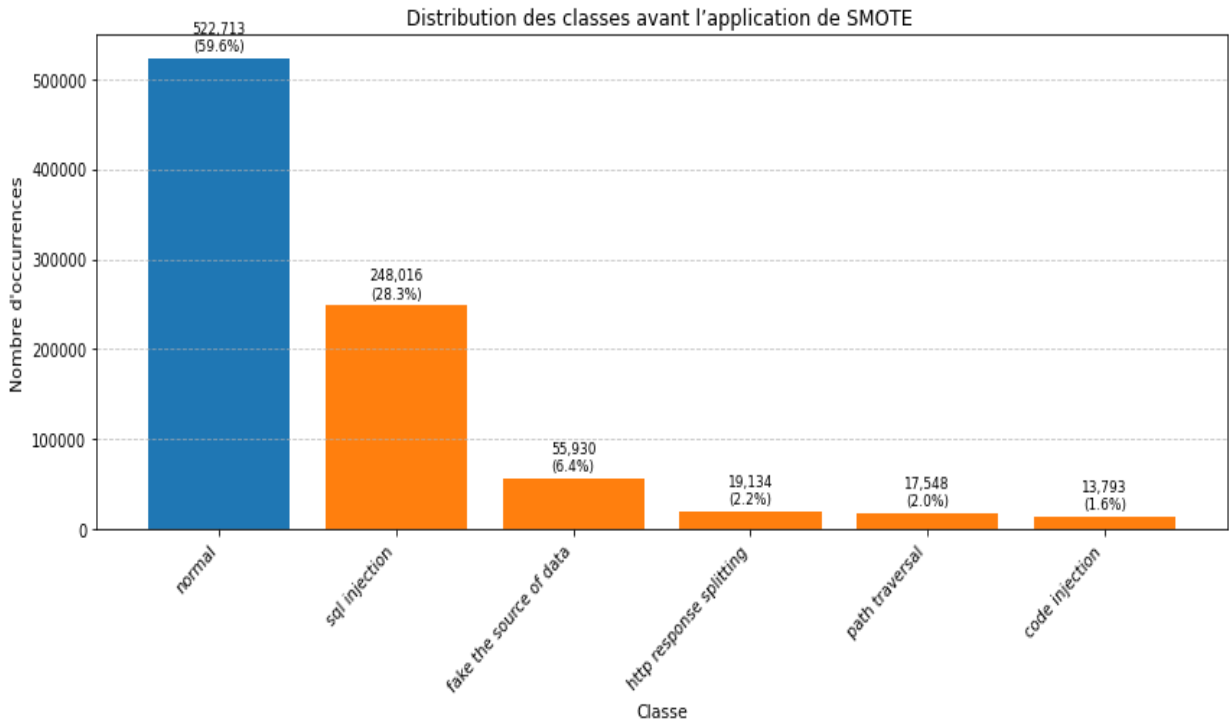

Figure 22 : Distribution des classes avant l'application de SMOTE

Après application la fonction SMOTE

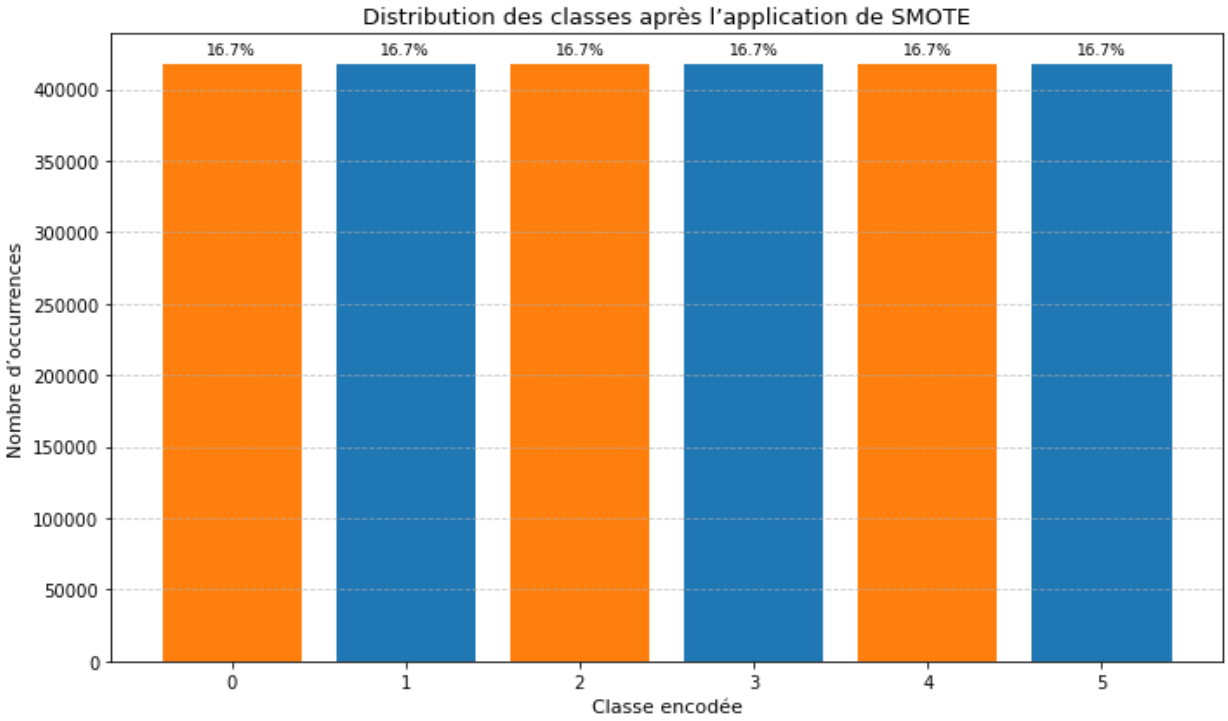

Figure 23 : Distribution des classes après l'application de SMOTE

Le graphique illustre une distribution équilibrée des classes, avec une part égale de 16,7 % pour chacune des six classes, ce qui indique que les données ont été traitées pour corriger un déséquilibre initial. Grâce à cette nouvelle répartition uniforme, le modèle d'apprentissage automatique pourra être entraîné de manière plus efficace et juste, en évitant les biais liés aux classes majoritaires.

5. Application des modèles d'apprentissage automatique sur le dataset

5.1. Choix des modèles

Après avoir terminé les étapes essentielles de prétraitement des données, incluant le nettoyage, la transformation, la sélection des caractéristiques, la gestion du déséquilibre des classes et la division du dataset, nous passons maintenant à l'étape cruciale en apprentissage automatique : l'application des modèles.

Cette phase vise à entraîner et évaluer différents algorithmes d'apprentissage automatique sur le dataset prétraité, dans le but de détecter efficacement les attaques dans les applications web. L'objectif principal est de comparer les performances de plusieurs modèles afin d'identifier celui qui offre les meilleurs résultats en termes accuracy et de F1-score, des

métriques de performance que nous détaillerons dans la section suivante.

Plusieurs algorithmes supervisés adaptés à la nature du problème (attaque vs normal) ont été choisis et appliqués sur le premier dataset **classification binaire** (Hybrid dataset : CSIC_HTTPParams), ainsi que sur un deuxième dataset **classification multiclasse** (SR-BH 2020), qui diffère légèrement en contenant des données pour la classe normale ainsi que pour différents types d'attaques.

Les algorithmes utilisés sont les suivants :

- Random Forest
- Logistic Regression
- LightGBM
- XGBoost

Nous avons retenu les modèles Random Forest, Logistic Regression, XGBoost et LightGBM en raison de leur efficacité reconnue dans les tâches de classification, notamment en sécurité des applications web**. Ils sont largement utilisés dans les études antérieures** et offrent des approches complémentaires. La régression logistique est simple et rapide, tandis que Random Forest améliore la précision grâce à l'utilisation de plusieurs arbres. XGBoost et LightGBM sont plus avancés et performants sur des données complexes.

Un autre facteur déterminant dans ce choix est que, parmi plusieurs modèles que nous avons testés au cours de notre étude, ces quatre-là ont systématiquement affiché des résultats supérieurs aux autres, en particulier en termes d'accuracy et de F1-score.

Chaque modèle sera entraîné sur 80 % des données d'entraînement, puis évalué sur les 20 % restantes du dataset de test. Cette étude comparative nous permettra de mesurer l'efficacité et la performance des modèles dans la détection et la classification précise sur ces différents datasets.

5.2 Mesures d'évaluation des Modèles de Classification

Avec la diversité des modèles d'apprentissage profond utilisés pour la détection des attaques web, et leur évolution continue grâce aux recherches dans ce domaine, on observe des différences notables dans leur structure et leur mode de fonctionnement. Chaque modèle adopte une approche spécifique pour traiter et analyser les données, ce qui lui confère des avantages particuliers dans différents contextes. Cette diversité rend l'évaluation de l'efficacité des

modèles plus complexe, notamment dans le cadre de la recherche constante d'amélioration des performances. Par conséquent, il devient essentiel d'utiliser des métriques d'évaluation standardisées permettant d'analyser objectivement les performances de chaque modèle et de les comparer entre eux pour identifier le plus adapté. Ces métriques permettent de mesurer la précision, l'efficacité ainsi que la capacité des modèles à être généralisé, en faisant un outil fondamental pour juger de leur pertinence dans des environnements et des conditions variés.

Tableau 12 : Éléments de la matrice de confusion

Concept	Définition	Résultat
TP - Vrais positifs	Cas positifs correctement prédits comme positifs	Détection correcte des attaques réelles
TN - Vrais négatifs	Cas négatifs correctement prédits comme négatifs	Exclusion correcte des demandes légitimes
FP - Faux positifs	Cas négatifs prédits à tort comme positifs	Alerte fausse - demande légitime classée comme attaque
FN - Faux négatifs	Cas positifs prédits à tort comme négatifs	Échec de détection - attaque réelle non identifiée

Avant d'aborder ces métriques, il était nécessaire de clarifier les concepts de base qui doivent être compris et maîtrisés au préalable, car ces métriques dépendent des concepts expliqués dans le tableau 1. Une fois ces bases établies, il devient plus facile de comprendre et d'interpréter les métriques de manière plus claire. Parmi ces métriques, il y en a certaines qui sont les plus utilisées dans les travaux précédents, et elles seront détaillées dans les paragraphes suivants :

L'exactitude (Accuracy) : L'exactitude mesure la proportion des prédictions correctes (positives et négatives) parmi l'ensemble des prédictions effectuées. (Liu & Lang, 2019)

$$\boldsymbol{Accuracy} = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN})}$$

Le rappel (Recall ou Sensitivity) : Le rappel mesure la capacité du modèle à détecter les vraies attaques (positifs réels). (Liu & Lang, 2019)

$$\boldsymbol{Recall} = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FN})}$$

La précision (Precision) : La précision mesure la proportion des cas réellement positifs parmi ceux que le modèle a prédits comme positifs. (Liu & Lang, 2019)

$$\mathbf{Precision} = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FP})}$$

Le F1-score : Le F1-score est la moyenne harmonique entre la précision et le rappel. Il donne un équilibre entre les deux métriques, utile en cas de classes déséquilibrées. (Liu & Lang, 2019)

$$\mathbf{F1-score} = \frac{2 \times (\mathrm{Precision} \times \mathrm{Recall})}{\mathrm{Precision} + \mathrm{Recall}}$$

5.3. Application des modèles

Dans cette section, nous présentons les différents modèles d'apprentissage automatique utilisés pour la détection des attaques web. Ces modèles, **sélectionnés pour leur efficacité et leur adaptabilité,** seront évalués sur leur capacité à classifier avec précision les requêtes malveillantes et normales, en mettant en lumière leurs performances respectives.

5.3.1 Random Forest

Les forêts aléatoires (Random Forests) sont une approche d'apprentissage automatique qui repose sur l'agrégation de nombreux arbres de décision générés aléatoirement. Introduite par Breiman, cette méthode tire parti de la diversité des arbres pour accroître la fiabilité et la précision des résultats, que ce soit pour des tâches de classification ou de régression. Elle se caractérise par sa robustesse face au surapprentissage, sa facilité d'utilisation et sa performance sur des données à haute dimension. (Simon , 2011)

- **Résultats Obtenus de classification binaire**

Les résultats de l'application du modèle Random Forest sur le dataset Hybrid dataset : CSIC_HTTPParams sont résumé comme suit :

Tableau 13: Résultats de modèle Random Forest en classification binaire

Caractéristiques	Accuracy	Type d'attaque	Precision	Recall	F1 Score
Random Forest	0.9921	0	0.93	0.98	0.95
		1	1.00	0.99	1.00

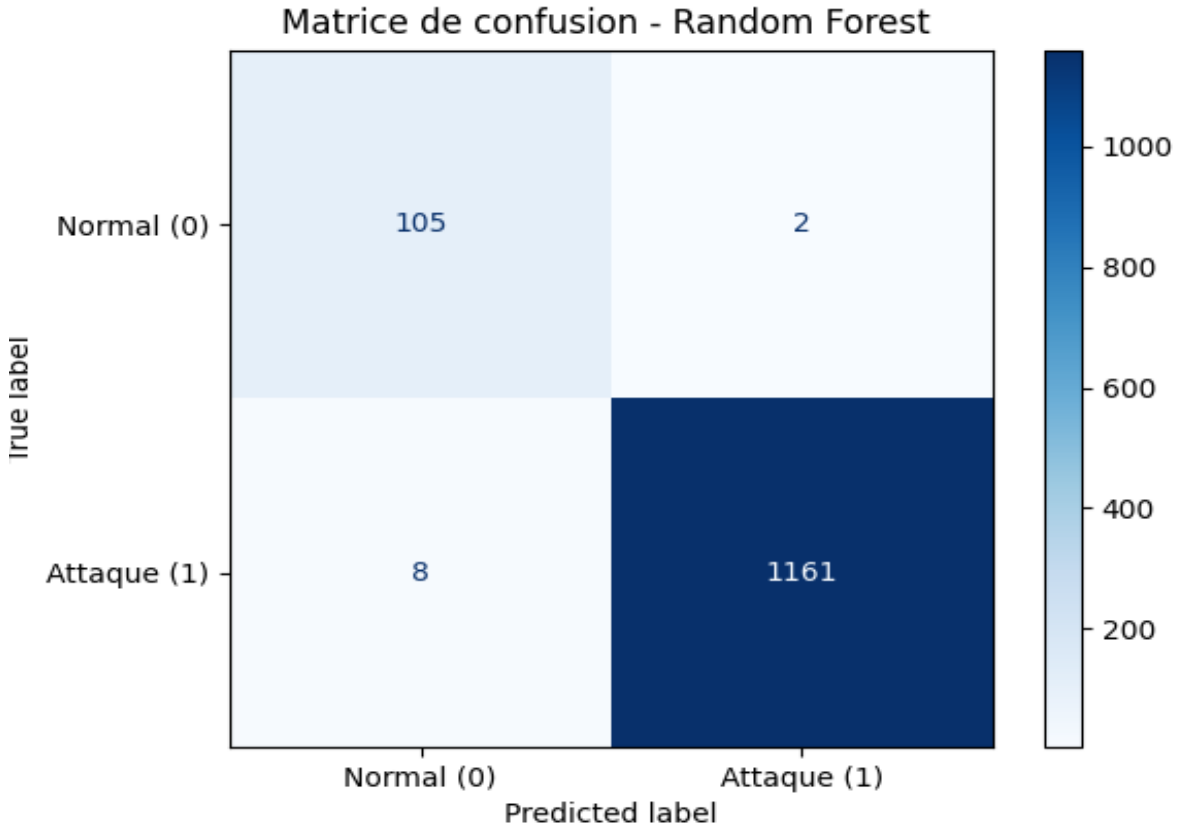

Figure 24 : Matrice de confusion Random Forest en classification binaire

À partir des résultats précédents, le modèle Random Forest a démontré une performance remarquable dans la classification binaire des requêtes web, avec une précision globale (Accuracy) de 0.9921 sur le premier dataset. Il parvient à bien distinguer entre les requêtes normales et malicieuses, affichant un bon équilibre entre les deux classes. Pour la classe "Normal", la précision est de 0.93 et le rappel de 0.98, ce qui reflète une faible incidence de faux positifs, tandis que pour la classe "Attaque", les scores sont presque parfaits avec une précision et un F1-score de 1.00, et un rappel de 0.99. Ces résultats sont corroborés par la matrice de confusion (Figure 24), où le modèle a correctement classé 105 échantillons normaux sur 107 et 1161 attaques sur 1169, ne commettant que peu d'erreurs. Ainsi, le modèle Random

Forest s'avère particulièrement efficace pour la détection des attaques dans le domaine de la sécurité des applications web.

- **Résultats Obtenus de de classification multiclasse**

Les résultats de l'application du modèle Random Forest sur le dataset SR-BH 2020 sont résumés ainsi :

Tableau 14 : Résultats du modèle RandomForest pour la classification multiclasse

Dataset		**SR-BH 2020**				
Métriques		**Accuracy**	**F1-score**	**Precision**	**Recall**	**F1-score**
RandomForest	**Code injection**	0,9236	0.9277	0.96	0.98	0.97
	Fake the source of data			0.99	1	0.99
	Http response splitting			0.55	0.81	0.66
	Normal			0.97	0.92	0.94
	Path traversal			0.48	0.88	0.62
	SQL injection			0.91	0.93	0,92

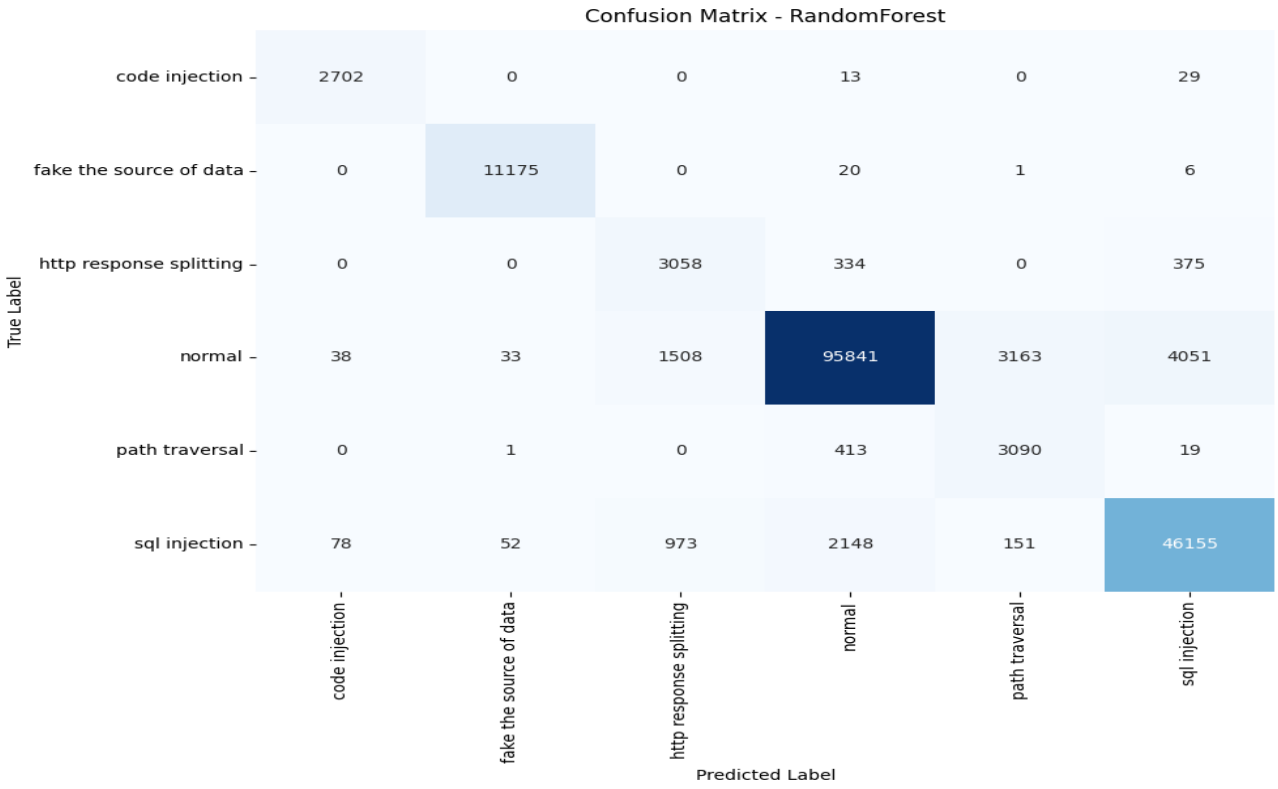

Figure 25 : Matrice de confusion Random Forest en classification multiclass

Selon les résultats précédemment obtenus, le modèle Random Forest a démontré de bonnes performances sur le dataset SR-BH 2020, avec une accuracy moyenne de 0.9236 et un F1-score de 0.9277. Les classes « Fake the source of data » (accuracy 0.99, F1-score 1.00), « Code injection » (accuracy 0.9236, F1-score 0.9277, precision 0.96, recall 0.98) et « Normal » (accuracy 0.97, F1-score 0.92, precision 0.94, recall 0.94) ont été bien reconnues. En revanche, des performances plus faibles ont été observées pour « Http response splitting » (accuracy 0.55, F1-score 0.81) et « Path traversal » (accuracy 0.48, F1-score 0.88). La matrice de confusion (Figure 25) confirme ces résultats, avec 95 841 échantillons de la classe « Normal » correctement classés, malgré quelques confusions avec les classes malveillantes. Le modèle reste néanmoins performant dans un contexte de classification multiclasse pour la sécurité des applications web.

5.3.2 Régression logistique

La régression logistique est une méthode d'analyse statistique utilisée pour modéliser la probabilité d'occurrence d'un événement binaire à partir d'un ensemble de variables

explicatives. Dans le contexte de la cybersécurité et de la détection des attaques web, elle permet de prédire la présence ou l'absence d'une attaque (par exemple : Attaque / Normal) en fonction de différentes caractéristiques extraites des requêtes HTTP ou du trafic réseau. Cette méthode est particulièrement utile lorsque la variable à prédire est dichotomique, et elle s'adapte bien aux données réelles, sans exiger de conditions strictes sur la distribution ou la variance. Grâce à ses capacités d'interprétation via les rapports de cote, elle permet d'identifier les facteurs les plus influents associés aux comportements suspects ou malveillants. (Desjardins, 2007)

- **Résultats obtenus de classification binaire**

Les résultats de l'application du modèle Logistic Regression sur le dataset Hybrid dataset : CSIC_HTTPParams sont résumé comme suit :

Tableau 15: Résultats de modèle Logistic Regression en classification binaire

Caractéristiques	Accuracy	Type d'attaque	Precision	Recall	F1 Score
Logistic Regression	0,9529	0	0.65	0.94	0.77
		1	0.99	0.95	0.97

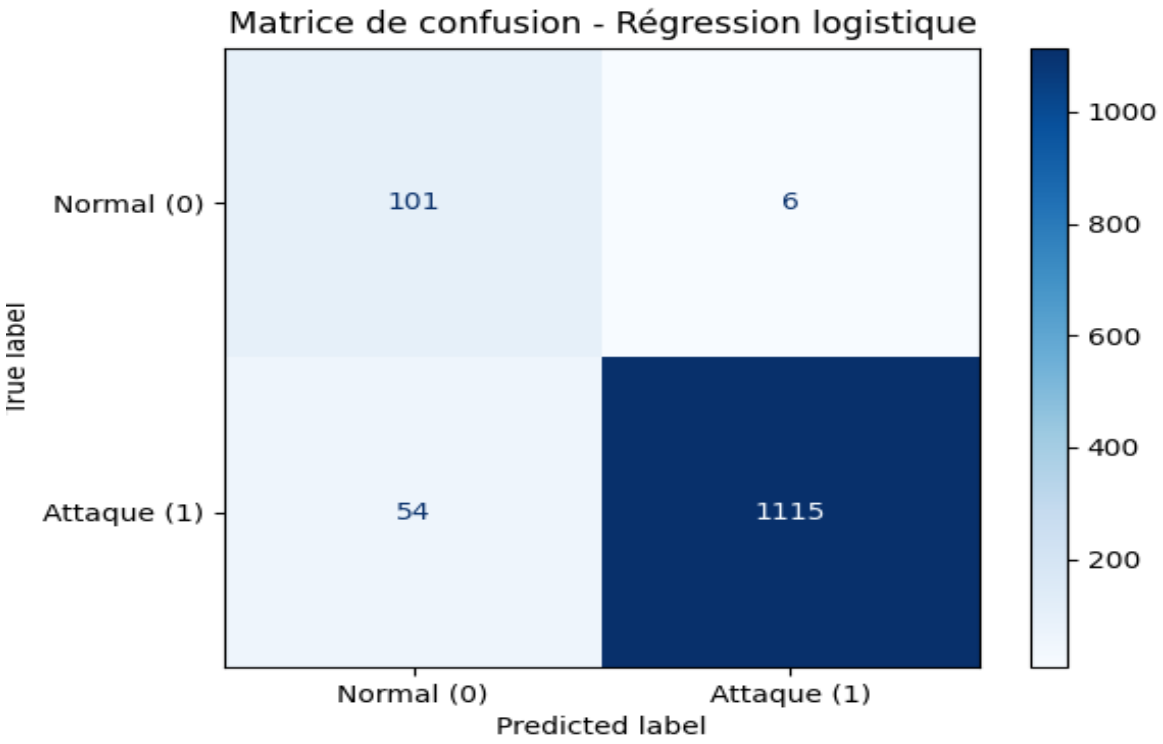

Figure 26 : Matrice de confusion Regression Logistique en classification binaire

D'après les résultats présentés, le modèle de régression logistique montre une accuracy globale de 0.9529, ce qui reflète une bonne performance générale. Pour la classe « Attaque » (1), les résultats sont excellents avec une précision de 0.99, un recall de 0.95 et un F1-score de 0.97, indiquant une grande efficacité dans la détection des requêtes malveillantes. En revanche, pour la classe « Normale » (0), bien que le recall atteigne 0.94, la précision chute à 0.65 en raison d'un nombre important de faux positifs, ce qui entraîne une baisse du F1-score à 0.77. La matrice de confusion (Figure 26) confirme cette tendance : le modèle excelle dans la détection des attaques, mais présente des difficultés à identifier correctement les requêtes légitimes.

- **Résultats obtenus de classification multiclasse**

  Les résultats de l'application du modèle Logistic Regression sur le dataset SR-BH 2020 sont résumé comme suit :

Tableau 16 : Résultats de modèle Logistic Regression en classification multiclasse

Dataset		**SR-BH 2020**				
Métriques		**Accuracy**	**F1-score**	**Precision**	**Recall**	**F1-score**
Regression Logistique	**Code injection**	0.5498	0.6241	0.05	0.07	0.06
	Fake the source of data			0.12	0,50	0.19
	Http response splitting			0.07	0.14	0.09
	Normal			0.99	0.71	0.83
	Path traversal			0.10	0.69	0.17
	SQL injection			0,75	0.27	0.39

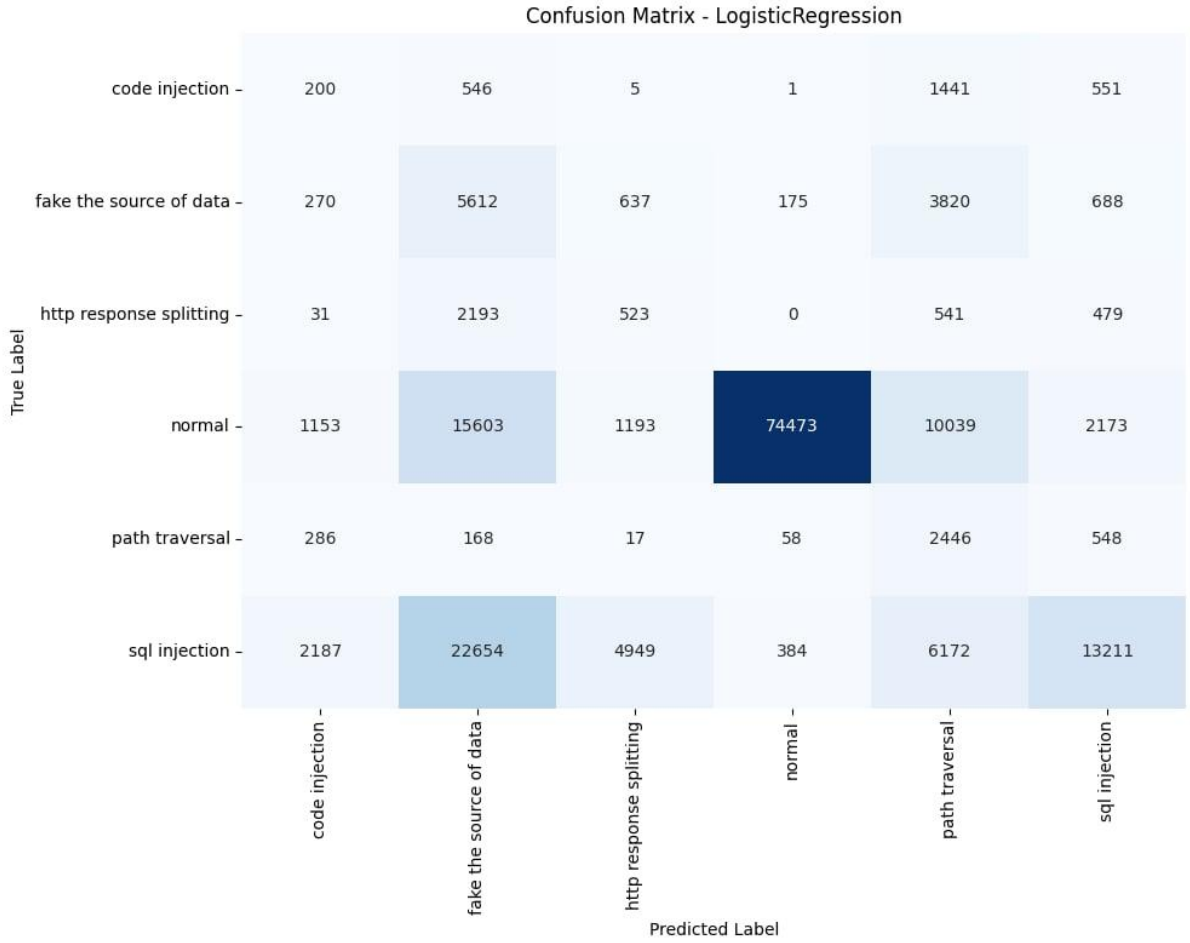

Figure 27 : Matrice de confusion Regression Logistique en classification multiclasse

Comme présenté dans la figure de la matrice de confusion et dans le tableau des métriques, le modèle de Régression Logistique présente des performances globalement faibles sur le dataset SR-BH 2020, avec une accuracy de 0.5498 et un F1-score global de 0.6241. Seule la classe « Normal » se distingue par de bons résultats, avec 74 473 instances correctement prédites, une précision de 0.99, un rappel de 0.71, et un F1-score de 0.83. La classe « SQL injection » montre également une certaine efficacité, avec un F1-score de 0.39. En revanche, les autres classes, telles que « Code injection », « Fake the source of data », « Http response splitting » et « Path traversal », enregistrent des scores très bas. Par exemple, « Code injection » n'a qu'une précision de 0.05 et un F1-score de 0.06. Ces faibles performances révèlent une forte confusion entre classes malveillantes, comme le montre la matrice, où les attaques sont souvent classées comme « normales » ou comme d'autres types d'attaques. Cela confirme que la Régression Logistique n'est pas adaptée pour distinguer correctement la majorité des

attaques, en dehors des requêtes normales.

5.3.3 LightGBM

C'est un algorithme d'apprentissage automatique basé sur le Gradient Boosting utilisant des arbres de décision, et il est utilisé pour résoudre des problèmes de classification et de prédiction. Cet algorithme construit des modèles puissants en assemblant progressivement plusieurs modèles faibles, en accordant une importance plus grande aux données difficiles à classer à chaque étape. Ce qui distingue LightGBM, c'est sa grande rapidité et sa capacité à gérer efficacement de grandes quantités de données, grâce à des techniques telles que la sélection des échantillons les plus influents et la fusion des caractéristiques afin de réduire la taille et d'améliorer la performance**.** (Sureda Riera et al, 2022)

- **Résultats obtenus de classification binaire**

Les résultats de l'application du modèle LightGBM sur le dataset Hybrid dataset : CSIC_HTTPParams sont résumé comme suit :

Tableau 17 : Résultats du modèle LightGBM pour la classification binaire

Caractéristiques	Accuracy	Type d'attaque	Precision	Recall	F1 Score
LightGBM	0.9945	0	0.95	0.99	0.97
		1	1.00	0.99	1.00

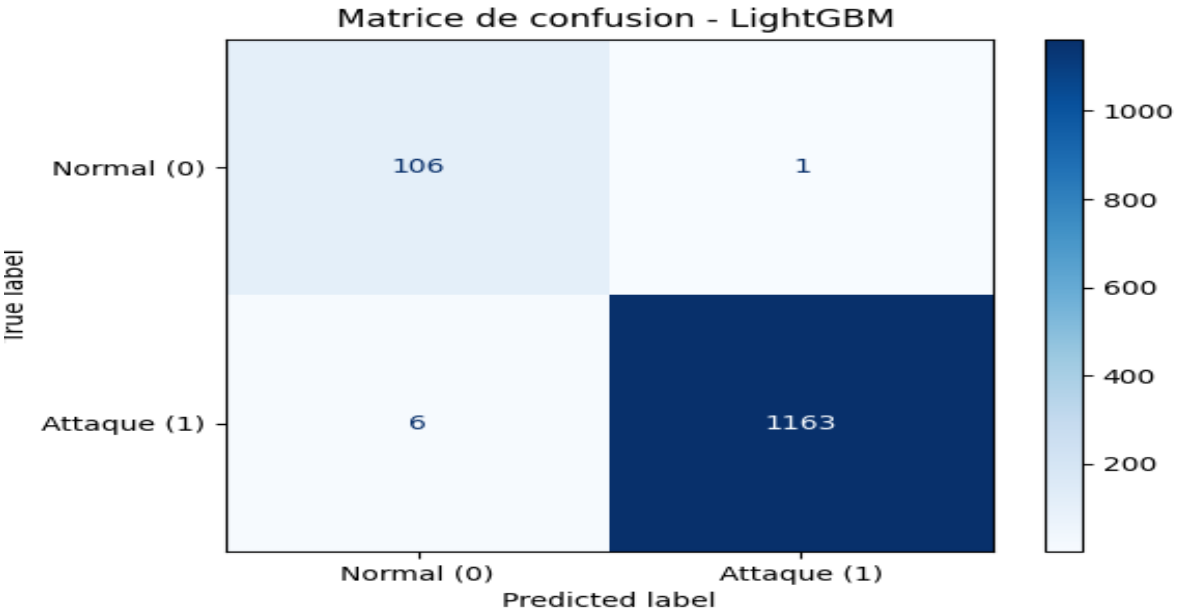

Figure 28 : Matrice de confusion LightGBM en classification binaire

Les résultats présentés dans le tableau et la figure indiquent que le modèle LightGBM présente des performances remarquables avec une accuracy de 99.45 %, appuyée par une matrice de confusion révélant très peu d'erreurs (1 faux positifs et 6 faux négatifs). Il atteint un F1-score de 0.97 pour les activités normales et de 1.00 pour les attaques, ce qui démontre une excellente capacité à détecter les intrusions tout en minimisant les fausses alertes. Ces résultats indiquent que le modèle est bien entraîné et parvient à généraliser efficacement, en maintenant un équilibre optimal entre sensibilité (rappel élevé) et spécificité (faible taux de faux positifs), ce qui est essentiel dans les systèmes de détection d'intrusions pour éviter à la fois les menaces non détectées et les alertes inutiles.

- **Résultats obtenus de classification multiclasse**

Les résultats de l'application du modèle LightGBM sur le dataset SR-BH 2020sont résumé comme suit :

Tableau 18 : Résultats du modèle LightGBM pour la classification multiclasse

Dataset		**SR-BH 2020**				
Métriques		**Accuracy**	**F1-score**	**Precision**	**Recall**	**F1-score**
LightGBM	**Code injection**	0,8957	0,9109	0.84	1	0.91
	Fake the source of data			0.97	1	0.99
	Http response splitting			0.30	0.98	0.46
	Normal			1	0.87	0.93
	Path traversal			0.45	0.99	0.62
	SQL injection			0.91	0.90	0.91

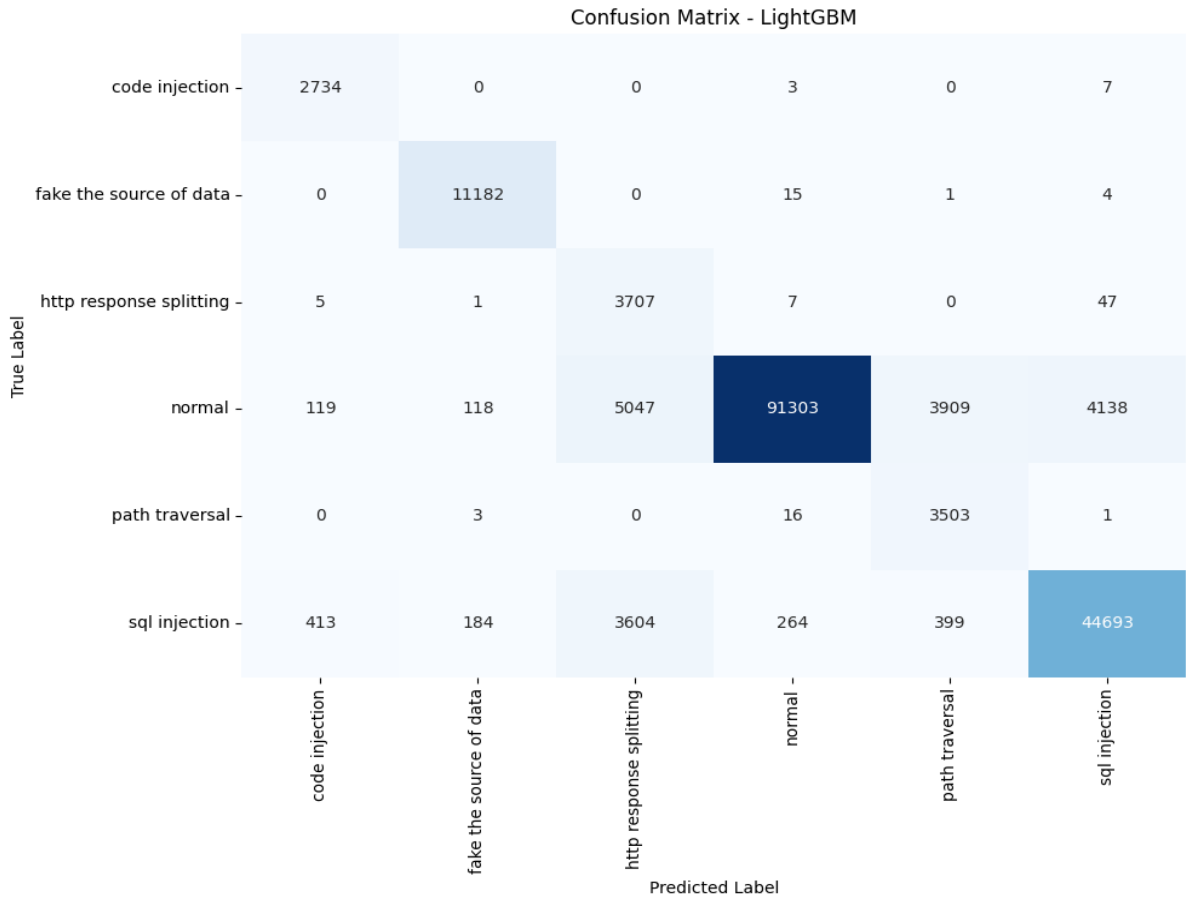

Figure 29 : Matrice de confusion LightGBM en classification multiclasse

À partir des résultats et de la matrice de confusion, le modèle LightGBM montre de très bonnes performances globales avec une accuracy de 89.57 %, notamment pour les attaques code

injection, fake the source of data et SQL injection (F1-score ≥ 0.90). Cependant, il présente des difficultés pour détecter http response splitting et path traversal, avec des F1-scores faibles (0.30 et 0.45), souvent confondues avec des requêtes normales. Le modèle reste néanmoins très fiable pour la classe normale, ce qui indique une bonne capacité à distinguer les comportements légitimes, mais une amélioration ciblée est nécessaire pour les classes minoritaires ou ambiguës.

5.3.4 XGBoost(eXtreme Gradient Boosting)

XGBoost est un algorithme de gradient boosting basé sur les arbres de décision, reconnu pour sa performance et son efficacité, notamment sur les données tabulaires. Il fonctionne en construisant une séquence de modèles où chaque modèle affine les prédictions en tenant compte des résultats précédents. Dès la première estimation, le modèle s'améliore progressivement en ajoutant des arbres successifs. XGBoost se distingue par sa rapidité d'exécution grâce au calcul parallèle, sa capacité à gérer automatiquement les valeurs manquantes, et son efficacité sans nécessiter une normalisation préalable des données. Ces caractéristiques en font un outil puissant pour les tâches de classification et de régression**.** (Hamour & Benhamdine, 2020)

- **Résultats obtenus de classification binaire**

Les résultats de l'application du modèle XGBoost sur le dataset Hybrid dataset : CSIC_HTTPParams sont résumé comme suit :

Tableau 19 : Résultats de modèle XGBoost en classification binaire

Caractéristiques	Accuracy	Type d'attaque	Precision	Recall	F1 Score
XGBoost	0,9945	0	0.95	0.99	0.97
		1	1.00	0.99	1.00

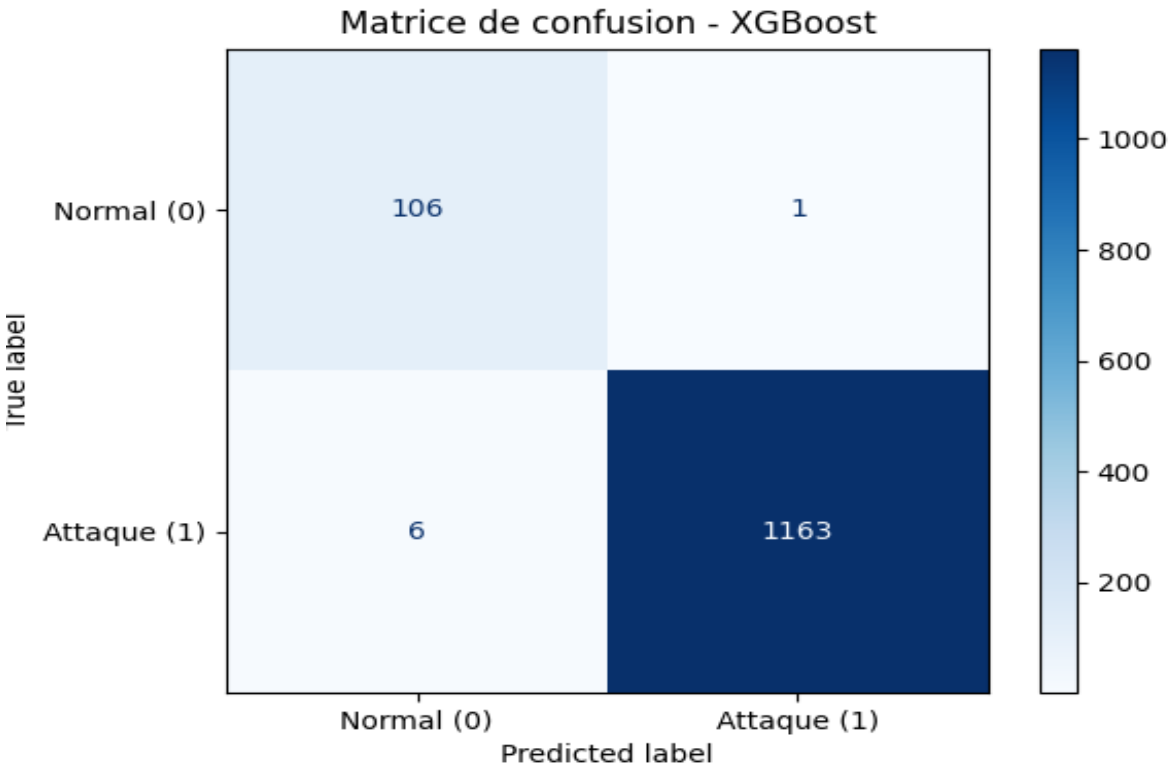

Figure 30 : Matrice de confusion XGBoost en classification binaire

À partir des résultats et de la figure de la matrice de confusion, le modèle XGBoost affiche d'excellentes performances avec une accuracy de 99,45 %. Il commet très peu d'erreurs, avec seulement 1 faux positif et 6 faux négatifs, ce qui traduit une grande précision dans la classification des connexions normales et des attaques. Le F1-score atteint 0,97 pour le trafic normal et 1,00 pour les attaques, illustrant une capacité remarquable à détecter les intrusions tout en minimisant les fausses alertes. Ce haut niveau de fiabilité rend le modèle particulièrement adapté aux systèmes de détection d'intrusion.

- **Résultats obtenus de classification multiclasse**

Les résultats de l'application du modèle XGBoost sur le dataset SR-BH 2020 sont résumé comme suit :

Tableau 20: Résultats de modèle XGBoost en classification multiclasse

Dataset		**SR-BH 2020**				
Métriques		**Accuracy**	**F1-score**	**Precision**	**Recall**	**F1-score**
XGBoost	**Code injection**	0.8986	0.9140	0.89	1	0.94
	Fake the source of data			0.98	1	0.99
	Http response splitting			0.30	0.98	0.46
	Normal			1	0.88	0.93
	Path traversal			0.45	0,99	0.62
	SQL injection			0.92	0,90	0.91

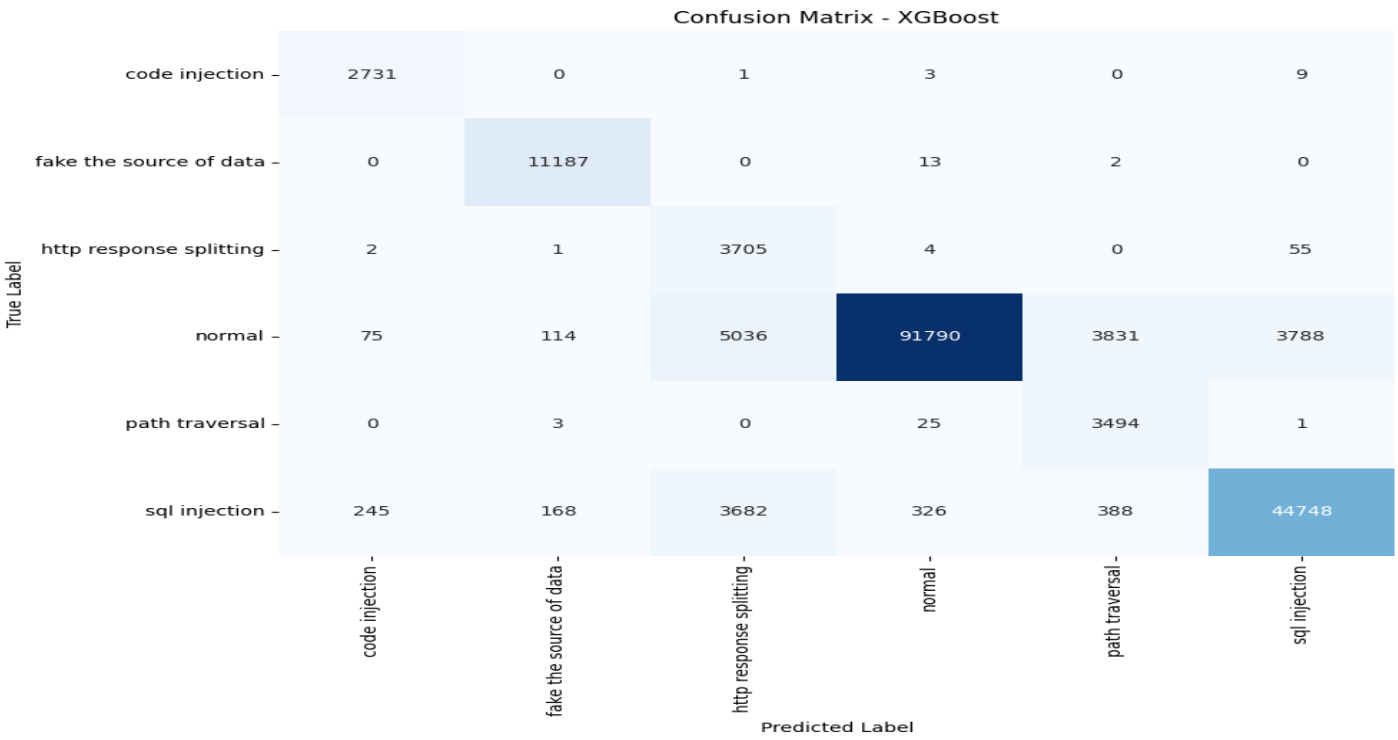

Figure 31 : Matrice de confusion XGBoost en classification multiclasse

À partir des résultats et de la matrice de confusion, le modèle XGBoost affiche de très bonnes performances globales avec une accuracy de 89.86 %, notamment pour les attaques code injection, fake the source of data et SQL injection (F1-score ≥ 0.90). Cependant, il rencontre des difficultés pour détecter http response splitting et path traversal, avec des F1-

scores faibles (0.30 et 0.45), souvent confondues avec des requêtes normales (par exemple, 3705 cas de http response splitting prédits comme normal). Le modèle reste très performant pour la classe normal (F1-score de 0.93), démontrant une bonne capacité à identifier les comportements légitimes, mais des améliorations sont nécessaires pour mieux gérer les classes minoritaires ou ambiguës.

5.4. Comparaison entre les résultats obtenus de classification binaire

Le tableau ci-dessous présente une comparaison des performances de plusieurs modèles de classification appliqués à la détection des attaques web.

Tableau 21 : Tableau comparatif des performances des modèles de classification binaire pour la détection des attaques

Caractéristiques	Accuracy	Type d'attaque	Precision	Recall	F1 Score
Random Forest	0.9921	0	0.93	0.98	0.95
		1	1.00	0.99	1.00
Logistic Regression	0.9529	0	0.65	0.94	0.77
		1	0.99	0.95	0.97
XGBoost	0.9945	0	0.95	0.99	0.97
		1	1.00	0.99	1.00
LightGBM	0.9945	0	0.95	0.99	0.97
		1	1.00	0.99	1.00

À partir de ce tableau, on observe que les modèles XGBoost et LightGBM présentent les meilleures performances parmi tous les modèles évalués. Ils affichent tous les deux une accuracy globale de 99,45 %, ce qui témoigne d'une excellente capacité de classification. En analysant plus en détail, on remarque qu'ils maintiennent un équilibre remarquable entre la précision, le rappel et le F1-score, aussi bien pour la classe 0 que pour la classe 1. Pour la classe 0, ils atteignent un F1-score de 0,97, supérieur aux autres modèles, tandis que pour la classe 1, ils obtiennent une précision parfaite de 1,00 et un F1-score également parfait de 1,00. Ces résultats indiquent que XGBoost et LightGBM sont capables de détecter efficacement les

attaques web tout en minimisant les erreurs de classification. Par conséquent, XGBoost et LightGBM peuvent être considérés comme les modèles les plus performants et les plus fiables pour cette tâche de détection des attaques web.

5.5. Comparaison des résultats obtenus sur dataset CSIC_HTTPParams et les études antérieures

Apres l'obtention des résultats des modèles sur les différents datasets, il est utile de voir les résultats d'autres auteurs sur les mêmes dataset obtenus par l'application d'un autre ensemble de modèles.

Nous nous intéressons dans l'interprétation des résultats du Hybrid dataset : CSIC_HTTPParams a la comparaison avec l'article (Shaheed & Kurdy, 2022). Suivant cet article, les auteurs ont appliqué plusieurs modèles sur le Hybrid dataset et qu'ils ont obtenus des résultats très proches de 96,4%. Cela nous invite à comparer nos résultats obtenus par l'application des modèles qu'on a utilisé sur le même dataset, les figures suivantes montre notre travail et le travail de (Shaheed & Kurdy, 2022)

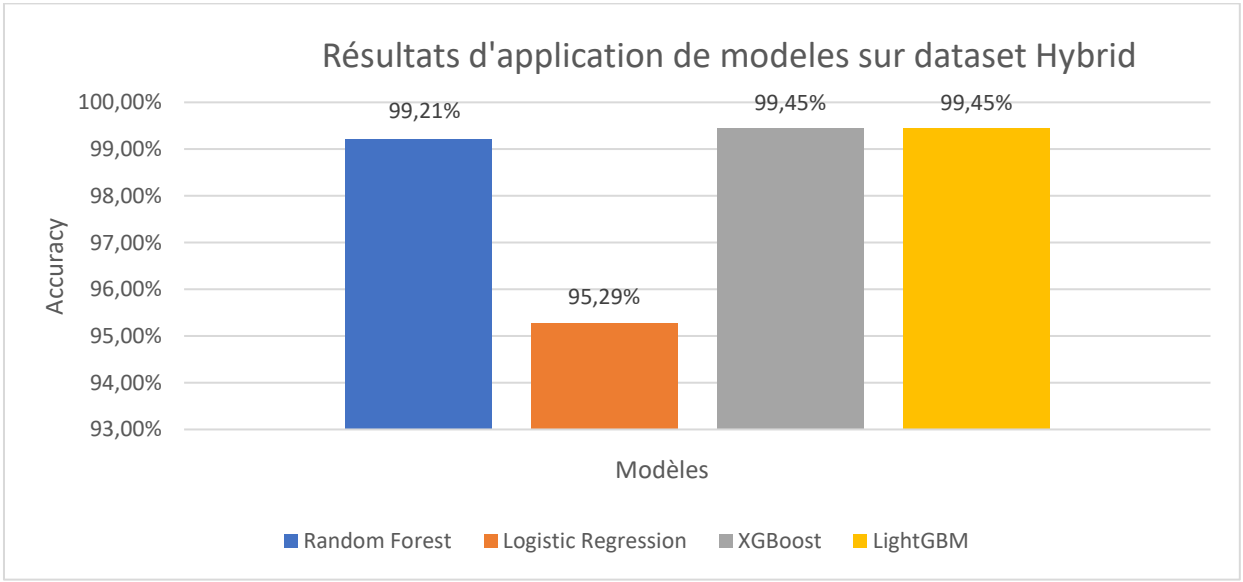

Figure 32: Résultats d'application de modèles sur dataset Hybrid

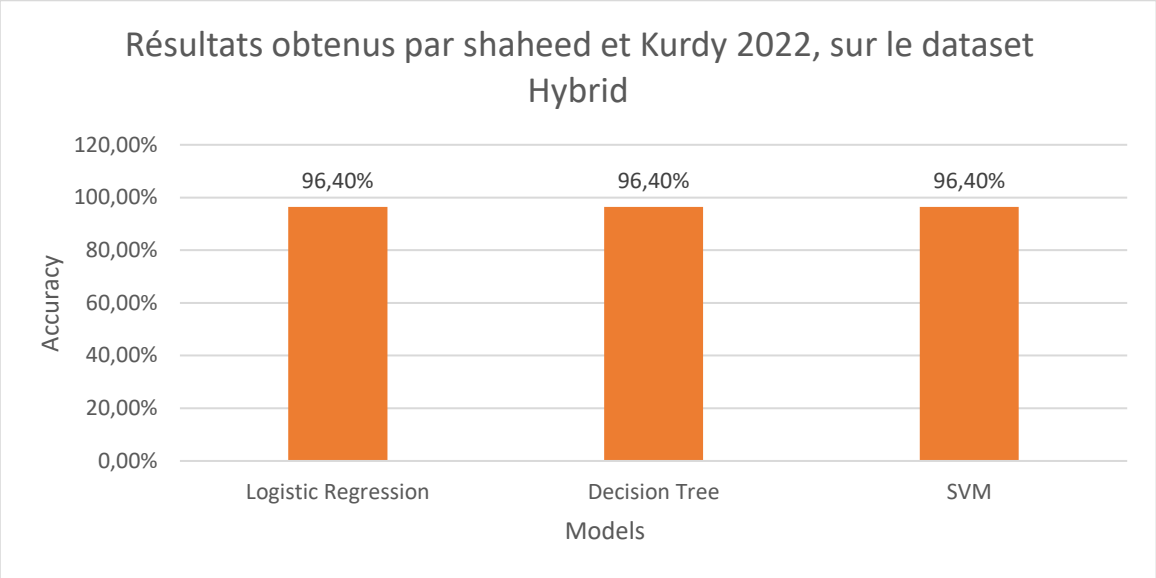

Figure 33 : Résultats obtenus par shaheed et Kurdy 2022, sur le dataset Hybrid

Le modèle qui est utilisé dans les deux travaux est Logistic regression dans lequel nous avons obtenu accuracy de 95,29%. Les auteurs ont obtenu 96,4% qui est traduite à une capacité de détection forte et proche de celle de l'article suscité. Ils affirment qu'ils ont réalisé des résultats similaires dans les autres modèles soit 96,4%. Cependant, nos résultats dans les autres modèles ont dépassé 99,2% d'accuracy chacun. **Dans le nettoyage de données, les auteurs ont supprimé les doublons et les valeurs manquantes, ils n'ont pas mentionné dans aucun cas l'utilisation de méthodes d'équilibrage de classes. Dans notre travail, nous avons exploité la méthode SMOTE pour rééquilibrer les classes. Cela explique l'écart d'accuracy entre leur travail et le nôtre.**

5.6. Comparaison entre les résultats des différents modèles de classification multiclasse

Le tableau ci-dessous présente une comparaison des performances de plusieurs modèles de classification, évalués par la précision, le rappel et le score F1, appliqués à la détection des attaques web.

Tableau 22 : Tableau comparatif des performances des modèles de classification multiclasse pour la détection des attaques

Dataset		**SR-BH 2020**				
Métriques		**Accuracy**	**F1-score**	**Precision**	**Recall**	**F1-score**
RandomForest	**Code injection**	0,9236	0.9277	0.96	0.98	0.97
	Fake the source of data			0.99	1	0.99
	Http response splitting			0.55	0.81	0.66
	Normal			0.97	0.92	0.94
	Path traversal			0.48	0.88	0.62
	SQL injection			0.91	0.93	0,92
Regression Logistique	**Code injection**	0.5498	0.6241	0.05	0.07	0.06
	Fake the source of data			0.12	0,50	0.19
	Http response splitting			0.07	0.14	0.09
	Normal			0.99	0.71	0.83
	Path traversal			0.10	0.69	0.17
	SQL injection			0,75	0.27	0.39
LightGBM	**Code injection**	0,8957	0,9109	0.84	1	0.91
	Fake the source of data			0.97	1	0.99
	Http response splitting			0.30	0.98	0.46
	Normal			1	0.87	0.93
	Path traversal			0.45	0.99	0.62
	SQL injection			0.91	0.90	0.91
XGBoost	**Code injection**	0.8986	0.9140	0.89	1	0.94
	Fake the source of data			0.98	1	0.99
	Http response splitting			0.30	0.98	0.46
	Normal			1	0.88	0.93
	Path traversal			0.45	0,99	0.62
	SQL injection			0.92	0,90	0.91

Le tableau présente une comparaison complète des performances de quatre modèles d'apprentissage automatique dans la détection de différents types d'attaques web.

Les résultats mettent clairement en évidence la supériorité des modèles 'LightGBM' et 'XGBoost', qui obtiennent des performances très élevées dans presque toutes les catégories d'attaques. Ces deux modèles affichent des précisions proches de 0,9 (respectivement 0,8957 et 0,8986), et des scores F1 très élevés, atteignant même la perfection (score de 1) dans certains cas. Cela démontre leur capacité remarquable à détecter les attaques avec fiabilité et cohérence. Le modèle 'RandomForest' obtient également de bons résultats, avec une précision globale de 0,9236. Toutefois, son score F1 varie fortement selon le type d'attaque. Par exemple, il chute à 0,62 dans certains cas, ce qui suggère que ce modèle peut rencontrer des difficultés face à des attaques plus complexes ou moins fréquentes.

En revanche, le modèle de 'Régression Logistique' montre des résultats nettement insuffisants. Il affiche une faible précision (0,5498), avec des scores F1 très bas pour la majorité des attaques, atteignant parfois des valeurs extrêmement faibles. Ces résultats indiquent que ce modèle n'est pas du tout adapté à la détection d'intrusions dans des contextes réels.

En conclusion, les modèles 'LightGBM' et 'XGBoost' se révèlent les plus performants et les plus fiables pour la détection d'attaques de sécurité web. Ils sont donc les mieux adaptés pour des applications réelles nécessitant une grande précision et un bon rappel. Le modèle 'Régression Logistique' est à éviter en raison de sa faible performance, tandis que 'RandomForest' peut représenter une option acceptable, mais avec certaines limites qu'il conviendrait de corriger ou d'optimiser.

5.7. Comparaison des résultats du dataset SR-BH 2020 avec les études antérieures

Après l'obtention des résultats des modèles sur différents datasets, il est pertinent d'examiner les résultats obtenus par d'autres auteurs sur des ensembles de données similaires en utilisant un autre ensemble de modèles. Nous nous intéressons particulièrement à l'interprétation des résultats du jeu de données SR-BH 2020, tel que présenté dans l'article (Sureda Riera et al T. , 2022). Selon cet article, les auteurs ont appliqué diverses combinaisons de modèles et d'algorithmes, notamment LightGBM et CatBoost avec des approches multi-label, en utilisant une nouvelle méthode d'extraction de caractéristiques basée sur les valeurs ASCII moyennes des requêtes web. Les résultats indiquent une supériorité notable du modèle Two-phase MultiOutput CatBoost dans différents scénarios de criticité. Cela nous incite à

comparer nos propres résultats, obtenus avec les modèles que nous avons utilisés sur le même jeu de données SR-BH 2020. Les figures suivantes illustrent notre travail ainsi que les résultats rapportés par (Sureda Riera et al T. , 2022).

Tableau 23 : Résultats des mesures du modèle LightGBM dans les deux travaux

Caractéristique	Light GBM (Sureda Riera et al T. , 2022)	Light GBM
Accuracy	0.8809	0,8957
Recall	0.9007	0.9566
Precision	0.8864	0.745

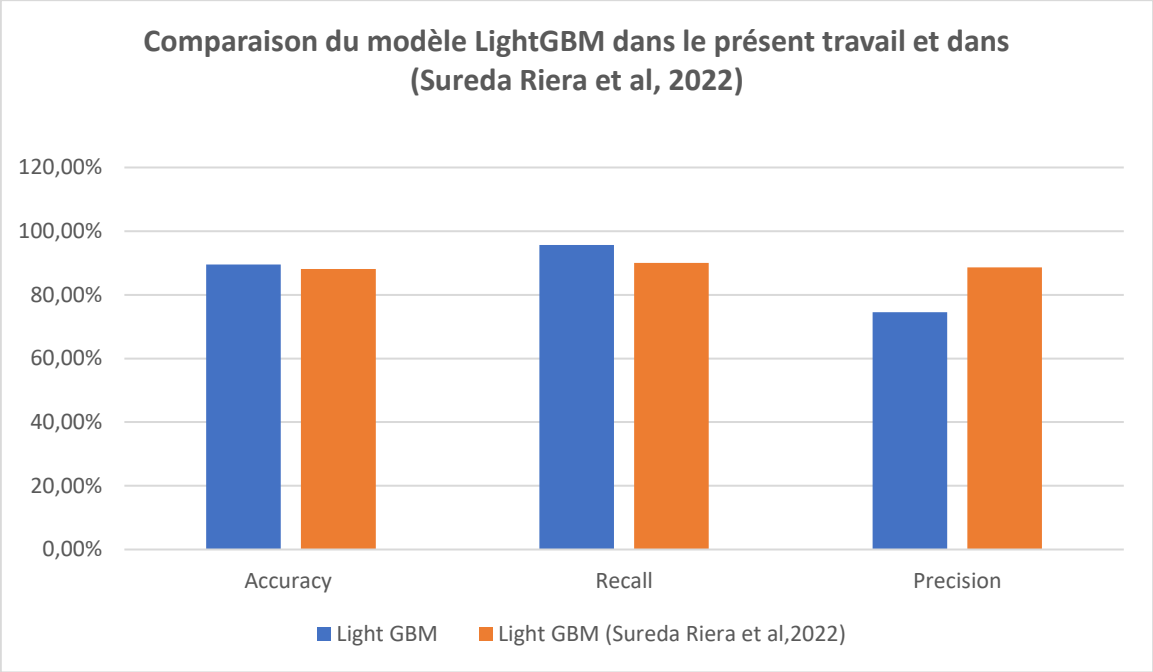

Figure 34 : Comparaison du modèle LightGBM dans le présent travail et dans (Sureda Riera et al, 2022)

Tableau 24 : Les meilleurs résultats obtenus sur les deux travaux

Caractéristique	RandomForest	CatBoost (Sureda Riera et al, 2022)
Accuracy	0,9236	0.8844
Recall	0.92	0.8882
Precision	0.81	0.9051

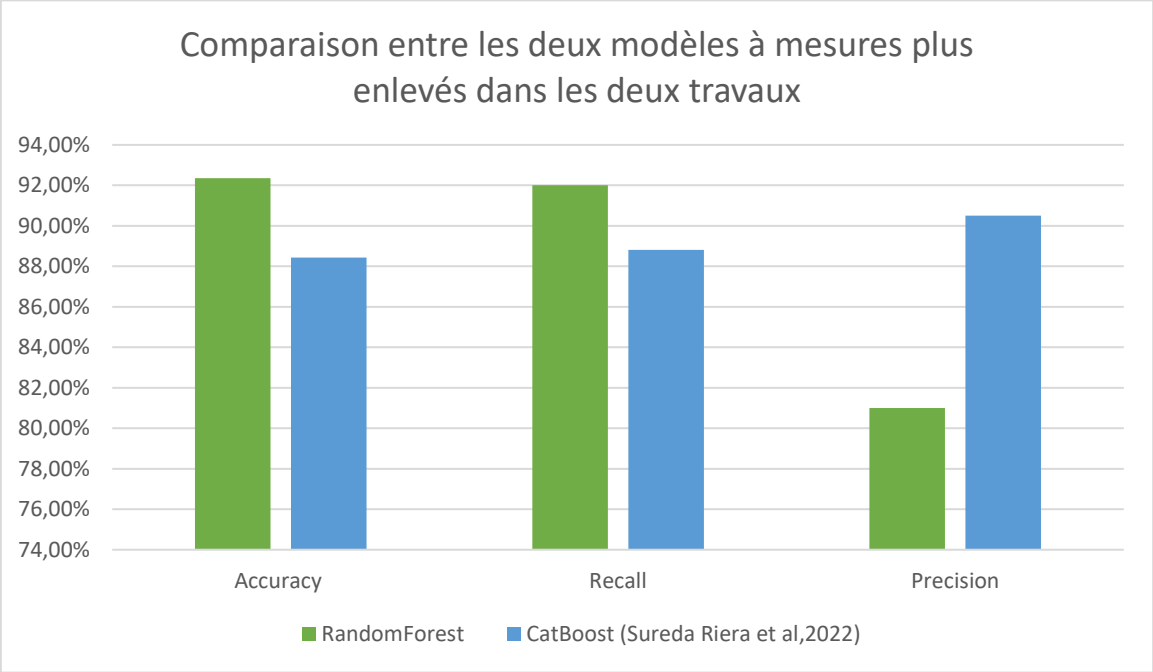

Figure 35 : Comparaison entre les deux modèles à mesures plus enlevés dans les deux travaux

A partir des figures et des tableaux précédents on observe que le modèle LightGBM du présent travail montre de meilleures performances en termes d'accuracy (0.8957 contre 0.8809) et de Recall (0.9566 contre 0.9007) par rapport à celui de LightGBM (Sureda Riera et al T., 2022). Cela indique que notre modèle détecte plus efficacement les attaques avec une meilleure classification globale. En revanche, la précision est plus faible (0.745 contre 0.8864), indiquant une augmentation des faux positifs. De même, lorsque l'on compare les performances de notre modèle RandomForest avec celles du modèle CatBoost (Sureda Riera et al, 2022), on constate que notre modèle obtient une accuracy plus élevée (0.9236 contre 0.8844) et un rappel

légèrement supérieur (0.92 contre 0.8882), ce qui reflète une meilleure capacité à détecter les attaques. Cependant, la précision de notre modèle est légèrement inférieure (0.81 contre 0.9051), ce qui peut indiquer une hausse modérée des faux positifs.
Ces résultats réfèrent à quelques des points positifs dans notre travail par rapport à (Sureda Riera et al, 2022), parmi lesquels :

- Capacité renforcée à détecter les attaques, même au prix d'un léger compromis sur la précision
- Utilisation de SMOTE pour traiter le déséquilibre des classes améliore de la détection des classes minoritaires.
- Choisir la classification multi classe a contribué à l'efficacité des modèles appliqués

Conclusion

Dans ce chapitre, nous avons présenté de manière détaillée la conception et la mise en œuvre d'un système de détection des attaques web fondé sur l'apprentissage automatique. Nous avons d'abord défini l'environnement de développement et les outils utilisés, avant d'expliquer les différentes étapes du prétraitement des données, appliquées successivement aux deux datasets : CSIC_HTTPParams et SR-BH 2020. Ces étapes comprenaient le nettoyage des données, la sélection des caractéristiques pertinentes, la gestion du déséquilibre des classes et la transformation des données.

Nous avons ensuite appliqué plusieurs algorithmes de classification, parmi lesquels Random Forest, Régression Logistique, LightGBM et XGBoost, dans des contextes de classification binaire et multiclasses. Une évaluation rigoureuse basée sur des métriques standards telles que la précision (accuracy), le rappel (recall), la précision (precision) et le F1-score nous a permis de comparer les performances des différents modèles. Les résultats obtenus, à savoir les valeurs élevées de ces métriques, relèvent du prétraitement et du passage de la classification multi label à la classification multi classe.

Chapitre 4 : Déploiement du modèle dans un environnement réel

Introduction

Dans le cadre de la validation du modèle que nous avons proposé pour la détection automatique des attaques ciblant les applications web, notamment les attaques de type SQL Injection, XSS, et Command Injection, nous avons jugé indispensable de procéder à une expérimentation dans un environnement réel.

Après les phases d'entraînement et de test réalisées dans un environnement contrôlé, il était nécessaire d'observer le comportement du modèle dans un contexte d'exécution plus représentatif des conditions réelles. Pour cela, nous avons mis en place un serveur web local (Apache2), utilisé comme plateforme d'expérimentation, afin de simuler des interactions dynamiques et évaluer la robustesse de notre modèle face à des requêtes authentiques, incluant des requêtes malveillantes.

Le modèle de détection utilisé dans cette expérimentation repose sur l'algorithme XGBoost, sélectionné à la suite d'une étude comparative menée sur le dataset CSIC_HTTPParams (classification binaire). Ce modèle a été retenu en raison de ses performances élevées observées lors des tests en environnement contrôlé, avec une accuracy atteignant 99.45 % et un F1-score de 0.97 pour la classe des attaques. Ces résultats ont motivé son intégration dans l'environnement réel présenté dans ce chapitre.

Cette démarche nous a permis non seulement d'évaluer les performances de détection en temps réel, mais aussi d'analyser les capacités d'intégration du modèle dans un système opérationnel. Ce chapitre présente ainsi l'environnement expérimental mis en place, les mécanismes de génération et d'analyse du trafic web.

1. Environnement de test

L'évaluation du modèle choisi a nécessité un cadre expérimental capable de reproduire des conditions proches d'un environnement réel. À cette fin, nous avons conçu un système de test local permettant la génération de trafic diversifié, l'observation du comportement du modèle face à des requêtes dynamiques, et l'analyse de ses performances dans un contexte réaliste.

1.1. Mise en place de l'environnement expérimental local

Afin de tester notre modèle dans un cadre contrôlé et sécurisé, nous avons opté pour l'utilisation de Windows Subsystem for Linux (WSL), permettant d'exécuter une distribution

Ubuntu directement sur un système Windows. Ce choix technique nous a permis de bénéficier de la flexibilité d'un environnement Linux tout en assurant une bonne compatibilité avec nos outils de développement.

Grâce à WSL, nous avons pu installer et configurer facilement les outils nécessaires à notre expérimentation, notamment le serveur web Apache2. Ce cadre local nous a également offert une gestion simplifiée des fichiers journaux (logs), essentiels pour l'analyse du comportement du modèle.

1.2. Configuration du serveur web Apache2

Dans l'environnement Ubuntu sous WSL, nous avons installé et configuré le serveur web Apache2, en raison de sa popularité dans les environnements de production, de sa fiabilité, et de sa compatibilité avec de nombreux outils d'analyse.

Nous avons adapté la configuration du serveur afin d'activer l'enregistrement automatique des requêtes HTTP via les fichiers de journalisation. Ces fichiers constituent une source précieuse pour suivre l'activité du serveur et analyser la réponse du modèle face aux différentes requêtes reçues. Nous avons également activé plusieurs modules nécessaires pour permettre l'exécution de scripts web et assurer une génération détaillée des journaux.

1.3. Présentation et exploitation du fichier access.log

Dans le cadre de notre expérimentation, le fichier access.log s'est révélé central. Ce journal, généré par Apache2, enregistre toutes les requêtes HTTP reçues par le serveur, ce qui nous permet de surveiller l'activité, d'identifier les requêtes suspectes, et d'alimenter notre système d'analyse avec des données réelles.

- **Emplacement du fichier access.log**

Le fichier access.log est généré automatiquement par Apache2 et se trouve dans le répertoire suivant :

```
/var/log/apache2/access.log
```

Dans notre environnement Ubuntu sous WSL, ce fichier était accessible localement, ce qui nous a permis de l'analyser aussi bien en temps réel qu'en différé.

- **Structure du fichier access.log**

Le fichier utilise généralement le format standard Common Log Format (CLF), où chaque ligne correspond à une requête HTTP. Un exemple typique d'entrée est présenté ci-dessous :

```
127.0.0.1 - - [13/Jun/2025:14:22:31 +0000] "GET /index.html
HTTP/1.1" 200 1024
```

Ce format contient les champs suivants :

Tableau 25 : Signification des champs dans le fichier access.log

Champ	Signification
127.0.0.1	Adresse IP du client
- -	Identifiants (utilisateur distant, authentifié)
[13/Jun/2025:14:22:31 +0000]	Horodatage de la requête
"GET /index.html HTTP/1.1"	Méthode, ressource demandée, protocole utilisé
200	Code de réponse HTTP (succès dans ce cas
1024	Taille de la réponse en octets

1.4. Installation des dépendances et du modèle de détection

Pour compléter la mise en place de notre environnement expérimental, nous avons procédé à l'installation des différentes dépendances nécessaires au bon fonctionnement de notre système de détection. Cette étape inclut à la fois les outils logiciels indispensables, tels que l'interpréteur Python et ses bibliothèques spécialisées, ainsi que le modèle d'apprentissage automatique entraîné en amont.

- **Installation de Python et des bibliothèques nécessaires**

Dans l'environnement Ubuntu installé via WSL, nous avons utilisé les commandes suivantes pour installer Python et les bibliothèques nécessaires :

```
sudo apt update
sudo apt install python3 python3-pip
pip3 install pandas xgboost watchdog
```

Les bibliothèques ci-dessus sont indispensables pour assurer le traitement des logs, l'extraction des caractéristiques et l'inférence par le modèle :

- Pandas : manipulation et structuration des données.
- xgboost : chargement et utilisation du modèle de classification .
- watchdog : pour la surveillance en temps réel du fichier access.log.

- **Chargement du modèle de détection**

Le modèle préalablement entraîné à l'aide de l'algorithme XGBoost est enregistré dans un fichier nommé xgboost_model.json. Il est exploité directement par le script principal afin de permettre une prédiction immédiate des requêtes interceptées, sans nécessiter de réentraînement.

Ce modèle a été sélectionné pour son efficacité démontrée sur le dataset CSIC_HTTPParams, et chargé ici pour assurer une détection rapide et fiable dans un environnement réel.

2. Architecture du système de détection

L'architecture du système de détection repose sur une série de composants interconnectés qui coopèrent de manière fluide afin de permettre l'identification automatique des attaques à partir des requêtes web enregistrées. Le schéma global de fonctionnement est illustré dans la figure suivante :

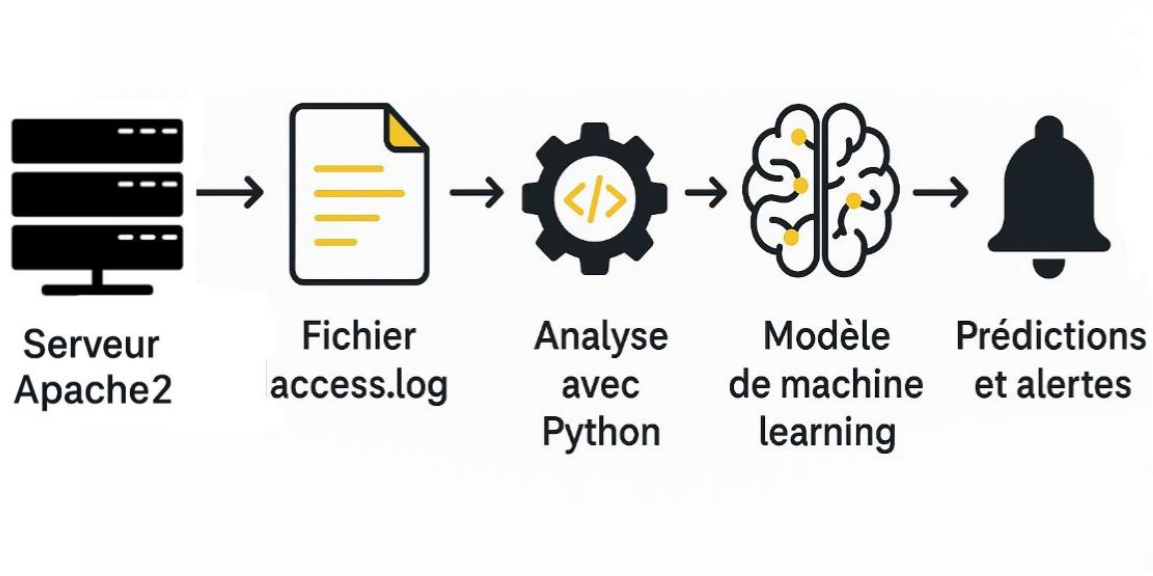

Figure 36 : Schéma d'analyse de la sécurité des applications web

Cette architecture s'articule autour de cinq composants principaux, décrits ci-dessous :

- **Serveur Apache2**

Il s'agit du serveur web responsable du traitement des requêtes HTTP. Toutes les requêtes entrantes, qu'elles soient légitimes ou malveillantes, sont traitées par Apache2 et enregistrées dans un fichier de journalisation (access.log).

- **Fichier access.log**

C'est le journal principal contenant les informations sur les requêtes reçues par le serveur. Chaque ligne du fichier correspond à une requête HTTP avec des données telles que l'IP de l'utilisateur, l'URL appelée, la méthode HTTP, le code de réponse, etc.

- **Analyse avec Python**

Un script Python surveille en temps réel ce fichier de log à l'aide de la bibliothèque watchdog. À chaque nouvelle ligne détectée, le script extrait les informations pertinentes (ex : URL, paramètres) et applique une série de transformations pour préparer les données.

- **Modèle d'apprentissage automatique**

Les caractéristiques extraites sont ensuite transmises à un modèle de classification entraîné en amont (par exemple, un modèle XGBoost). Ce dernier prédit si la requête analysée correspond à un comportement normal ou à une tentative d'attaque (SQLi, XSS, etc.).

- **Prédictions et alertes**

En fonction du résultat du modèle, une alerte est générée (sous forme d'un message coloré dans la console, d'une entrée dans un fichier CSV). Cela permet une surveillance proactive et une traçabilité des incidents détectés.

3. Pipeline de traitement en temps réel

Comme décrit dans la figure 36, la mise en œuvre opérationnelle du système se traduit par un pipeline de traitement en temps réel structuré en quatre étapes automatiques, assurant la surveillance des logs Apache, l'extraction des features, la classification via XGBoost, puis la journalisation et l'alerte en cas d'attaque ; nous détaillerons chaque opération dans les sous-sections suivantes**.**

3.1. Lecture continue du fichier access.log

Le processus commence par une surveillance permanente du fichier access.log généré par le serveur Apache. À l'aide du module watchdog, notre script Python détecte automatiquement toute nouvelle ligne ajoutée à ce fichier. Lorsqu'une requête HTTP est enregistrée (ex. : suite à une visite d'un utilisateur ou d'un robot), le script lit uniquement la nouvelle ligne sans relire tout le fichier, ce qui garantit un traitement efficace en temps réel.Chaque ligne est ensuite transmise à la fonction principale pour un traitement immédiat.

3.2 Extraction des caractéristiques (features)

Dès qu'une nouvelle requête est capturée, le système extrait plusieurs caractéristiques essentielles à partir du contenu du payload.

Ces caractéristiques sont utilisées comme entrée du modèle XGBoost, et permettent de différencier les requêtes normales des requêtes malveillantes.

Les features extraites sont :

- **payload_len** : longueur totale du contenu (query string) de la requête.
- **Alpha** : pourcentage de caractères alphanumériques (lettres/chiffres/espaces).
- **non_alpha** : pourcentage de caractères spéciaux ou symboles.
- **attack_feature** : un score pondéré basé sur :
  - La présence de chemins suspects (/admin, /.env, etc.).
  - Les expressions régulières représentant des patterns d'attaques (XSS, SQLi, etc..).
  - Le taux de symboles spéciaux dans la chaîne.
  - Ces valeurs sont structurées sous forme d'un vecteur pandas DataFrame, puis envoyées pour la prédiction.

3.3 Passage au modèle pour prédiction

Une fois les caractéristiques extraites, elles sont passées au modèle XGBoost, chargé à partir du fichier xgboost_model2.json.
Le modèle prédit :

- **Label** : une valeur binaire (1 pour attaque, 0 pour normal).
- **Probability** : la probabilité associée à cette classification (exprimée en pourcentage de confiance).

3.4. Journalisation et alerte en cas d'attaque détectée

À l'issue de la prédiction, les données sont enregistrées dans différents fichiers pour assurer une traçabilité complète :

- raw_requests.log : contient la requête brute extraite du access.log (IP, date, méthode, URL, etc..).
- features_results.csv : enregistre les caractéristiques calculées, la prédiction, la probabilité, et l'horodatage.
- attacks_detected.csv : n'est mis à jour que si une attaque est détectée, avec :
  - La requête encodée en base64.
  - Les types d'attaque détectés (XSS, SQLi, etc..).
  - Le score d'attaque.
  - Le timestamp complet.

En plus, un message d'alerte est affiché dans le terminal (en rouge), résumant toutes les informations pertinentes de l'attaque détectée (longueur, alpha, type..).

4. Exemple de fonctionnement

Pour illustrer concrètement le fonctionnement de notre système, nous avons appliqué un exemple pratique basé sur une attaque de type XSS, en suivant toutes les étapes de traitement, depuis la requête jusqu'à sa détection et journalisation.

4.1. Génération de la requête malveillante

Pour simuler une attaque de type XSS, nous avons utilisé l'outil curl afin d'envoyer une requête HTTP contenant un script JavaScript de type attaque.

```
curl "http://localhost/?q=<script>alert('xss')</script>"
```

Cette requête est automatiquement enregistrée par Apache dans son fichier de log, ce qui entraîne automatiquement son traitement par notre système de détection.

4.2 Analyse de la requête via le script Python

Comme mentionné précédemment, dès qu'un changement est détecté dans le fichier access.log, notre script Python se déclenche automatiquement. Il lit la ligne nouvellement ajoutée, puis extrait les caractéristiques pertinentes du payload :

- Longueur du payload : 31 caractères
- Pourcentage de caractères alphabétiques (alpha) : 67.7 %
- Pourcentage de caractères non alphabétiques (non_alpha) : 32.3 %
- Score de suspicion calculé (attack_feature) : 800.0

Ces données sont ensuite transmises au modèle d'apprentissage automatique pour effectuer la prédiction de la nature de la requête.

4.3. Prédiction par le modèle

Une fois les caractéristiques extraites, elles sont transmises au modèle XGBoost, chargé depuis le fichier xgboost_model.json.Ce modèle retourne une étiquette binaire (1 pour attaque,

0 pour normal), accompagnée d'une probabilité exprimant le degré de certitude.

Dans notre exemple, le modèle a classé la requête comme attaque (1) avec une probabilité de 100.00 %. Cette décision repose uniquement sur les caractéristiques extraites (payload_len, alpha, non_alpha, attack_feature)

4.4. Journalisation et capture

Après la phase de prédiction, le système enregistre automatiquement les résultats pour assurer une traçabilité complète.

- La requête interceptée est d'abord enregistrée telle quelle dans le fichier raw_requests.log :

```
   ::1 - - [14/Jun/2025:18:21:37 +0200] "GET
/?q=%3Cscript%3Ealert(%27XSS%27)%3C/script HTTP/1.1" 200 3460 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36 OPR/119.0.0.0"
```

- Les caractéristiques extraites ainsi que le résultat du modèle sont enregistrées dans features_results.csv :

```
 2025-06-14
16:04:43,GET,q=<script>alert('xss')</script>,31,67.74193548387096,32.
25806451612904,800,1,0.9999767541885376,xss
```

Une alerte est aussi affichée sur la console :

```
Attaque détectée !
• Heure    : 2025-06-14 16:04:43
• Requête  : q=<script>alert('xss')</script>
• Détails  : longueur (payload_len)=31, ALPHA=67.7%, SPÉCIAUX
(non_alpha)=32.3%
• Score attaque (attack_feature): 800.0
• Type(s) d'attaque : xss
• Probabilité : 100.00%
```

Finalement, les requêtes jugées malveillantes sont aussi archivées dans le fichier attacks_detected.csv, permettant une analyse ou un réentraînement ultérieur du modèle.

5. Mise à l'épreuve du système de détection choisi dans différents scénarios

Dans le cadre de l'évaluation du modèle dans un environnement réel, plusieurs requêtes HTTP ont été soumises au serveur web afin d'analyser la capacité du système à distinguer entre un trafic normal et des attaques de différentes natures (XSS, SQL Injection, Command Injection, Path Traversal).

Le tableau suivant présente un échantillon de ces requêtes, regroupées par type, avec le verdict du système (Normal ou Attaque) pour chaque cas. Ces résultats confirment l'efficacité du pipeline de détection en temps réel et sa réactivité face aux menaces.

Tableau 26 : Résultats de la détection pour différents types de requêtes HTTP

Type de trafic	**Requête HTTP**	**Résultat**		
		Prédictions	Probabilité	Type
Normal	" http://localhost/contact "	0	6.72%	Normal
	"http://localhost/images/logo.png "	0	6.72%	Normal
	" http://localhost/api/data?id=100 "	0	0.23%	Normal
	" http://localhost/blog/post?id=42 "	0	0.55%	Normal
	" http://localhost/?feedback=good+service "	1	65.55%	Attaque
	" http://localhost/?lang=en&theme=light "	1	87.99%	Attaque
	" http://localhost/?q=id=123&name=John "	0	48.95%	Normal

Attaque	XSS	"http://localhost/?q=<script>alert('XSS') </script>"	1	100%	XSS
		"http://localhost/?img=<img src=x onerror=alert (1)>"	1	100%	XSS
		" http://localhost/?q=<svg/onload=alert (1)>"	1	99.81%	Attaque
		" http://localhost/?q=javascript:alert('xss')"	1	100%	XSS
	SQL Injection	" http://localhost/?id=1' OR '1'='1 "	1	69.81%	SQL
		"http://localhost/?search='; DROP TABLE users; -- "	1	100%	SQL
		" http://localhost/?q=SELECT * FROM users WHERE name='admin "	1	100%	SQL
		" http://localhost/?user=admin'-- "	1	96.67%	SQL
	Command Injection	" http://localhost/?cmd=ls|whoami "	1	98.85%	CMD
		" http://localhost/?input=|cat /etc/passwd "	1	100%	Path,CMD
		" http://localhost/?run=; ping -c 4 evil.com "	1	66.42%	Attaque
		" http://localhost/?exec=&& netstat -an "	1	100%	SQL,CMD
	Path Traversal	"http://localhost/?file=../../../../etc/passwd"	1	99.94%	Path
		"http://localhost/?path=..\\..\\..\\..\\boot.ini "	1	99.94%	Path
		"http://localhost/?doc=../../../../var/log/apache2/access.log "	1	99.97%	Path
		"http://localhost/?page=../../../windows/win.ini "	1	99.59%	Path

À partir du tableau 26, on remarque que le système de détection est globalement capable de faire la distinction entre les requêtes normales et les attaques. Les requêtes légitimes sont généralement classées avec une faible probabilité d'attaque, ce qui montre que le modèle limite bien les faux positifs. En revanche, les attaques comme le XSS, le SQL Injection ou encore le

Path Traversal sont détectées avec une très grande précision, atteignant parfois 100 % de probabilité. Cela prouve l'efficacité du modèle pour identifier les menaces en temps réel. On note cependant quelques cas où des requêtes non malveillantes ont été considérées comme suspectes, ce qui indique une certaine sensibilité du système à des structures spécifiques. Malgré cela, les résultats restent très satisfaisants et démontrent la fiabilité du pipeline dans un contexte réel.

Conclusion

L'expérimentation présentée dans ce chapitre nous a permis de tester notre modèle de détection dans un environnement réel simulé. En utilisant un serveur Apache2 local, nous avons généré un trafic web varié, comprenant des requêtes normales ainsi que différentes formes d'attaques telles que XSS, SQL Injection et Command Injection.

Le modèle a montré une bonne capacité à détecter les attaques tout en limitant les faux positifs. Les résultats obtenus en situation réelle sont cohérents avec ceux des phases de test, ce qui confirme la stabilité et la fiabilité du modèle, même en dehors d'un environnement contrôlé. Le système mis en place, reposant sur la lecture en temps réel du fichier access.log, l'extraction automatique des caractéristiques et la classification immédiate, s'est révélé rapide, efficace et simple à intégrer.

Cette expérience démontre que notre approche est solide et adaptable, avec un potentiel d'utilisation dans des environnements plus vastes, tels que les pare-feux applicatifs ou les systèmes de surveillance réseau.

Conclusion générale

Conclusion générale

Avec l'évolution rapide du numérique et l'expansion massive des services en ligne, les applications web sont de plus en plus exposées à un large éventail de cyberattaques.

Dans ce contexte, la sécurité des applications web est devenue un enjeu majeur, nécessitant des solutions de défense plus intelligentes et adaptatives. Dans ce mémoire, nous avons tout d'abord présenté les principales vulnérabilités affectant les applications web, telles que les injections SQL, les attaques XSS, CSRF ou encore les attaques par force brute, tout en soulignant les limites des solutions traditionnelles. Par la suite, nous avons exploré l'apport de l'apprentissage automatique dans le domaine de la cybersécurité, en analysant plusieurs datasets de référence et en décrivant les étapes essentielles de prétraitement des données.Sur le plan pratique, nous avons appliqué certains modèles d'apprentissage automatique afin d'évaluer leur efficacité dans la détection des attaques web. Pour cela, deux datasets (CSIC_HTTPParams et SR-BH 2020) ont été utilisés, et diverses étapes ont été réalisées, telles que le nettoyage des données, la sélection des caractéristiques pertinentes et l'évaluation des performances des modèles. Les résultats ont démontré la pertinence des modèles Random Forest, LightGBM et XGBoost, notamment en classification multiclasse et multi-étiquettes, avec un bon équilibre entre précision et capacité de détection.

Dans un quatrième chapitre, nous avons mené une expérimentation réelle dans un environnement local, en déployant le modèle de détection sur un serveur local configuré avec Apache. L'analyse des fichiers de journalisation (access.log) nous a permis de tester l'efficacité du système face à des requêtes HTTP simulées, dans des conditions proches d'un contexte réel d'utilisation.

De manière générale, les résultats de ce travail démontrent l'efficacité de l'apprentissage automatique dans le renforcement des systèmes de détection et la sécurisation des applications web. Toutefois, il demeure nécessaire de poursuivre les efforts de recherche et de développement afin d'améliorer les performances en conditions réelles, d'élargir les capacités de détection en temps réel, de faire face aux attaques complexes et inconnues, tout en assurant une intégration fluide avec différents environnements et systèmes. Une piste intéressante pour les travaux futurs serait d'ajouter un mécanisme de réponse active capable de bloquer automatiquement les requêtes malveillantes identifiées, afin de renforcer la protection en temps réel des applications web.

Références

Ranjit, P., & Borah, S. (2018, Janvier). A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *Engineering & Technology*. Retrieved Juin 19, 2025, from https://www.researchgate.net/publication/329045441

Akrout, R. (2012). Analyse de vulnérabilités et évaluation de systèmes de détection d'intrusions pour les applications Web (Thèse de doctorat,Institut National des Sciences Appliquées de Toulouse). 142. Récupéré sur https://theses.hal.science/tel-00782565v1

Bhowmick, S. (2014). Command injection/shell injection. Springs. Retrieved Mai 24, 2025, from https://www.academia.edu/34168837/_Command_Injection

Biswas, S., Sajal, M., Afrin, T., Bhuiyan, T., & Hassan, M. (2018). A Study on Remote Code ExecutionVulnerability in Web Applications. *International Conference on Cyber Security and Computer Science (ICONCS'18)*, (p. 8). Turkey.

Boin, C. (2023). Détection d'attaques DDoS dans le contexte d'un fournisseur cloud de grande envergure (Thèse de doctorat,Université de Lille). 137. Récupéré sur https://hal.science/tel-04397580v1

BOUGHAZI, M., & LAKHAL , A. (2017, Juin). Attaque de l'homme du milieu dans les réseaux sociaux 4G(Mémoire de master ,universitéde Guelma). 72. Récupéré sur Kaspersky: https://dspace.univ-guelma.dz/jspui/bitstream/123456789/721/1/Boughazi%20%26%20Lakhal..pdf

Centre canadien pour la cybersécurité. (2024). *Se protéger des attaques par déni de service distribué (DDoS).* Centre canadien pour la cybersécurité, Ottawa. Consulté le Mai 23, 2025, sur (ITSM.80.110)

Desjardins, J. (2007). L'analyse de régression logistique. 7. Récupéré sur https://www.researchgate.net/publication/49619407_L%27analyse_de_regression_logistique

Elhanashi, A. e. (2023). *Lecture Notes in Electrical Engineering* ·. Retrieved Mai 24, 2025, from https://www.researchgate.net/publication/370384673

Gharibeh , S., Shatha , M., & Hassan , N. (2020). Classification on Web Application Requests. In IEEE (Ed.), *International Conference on Information and Communication Systems (ICICS)*, (p. 5). Retrieved Mai 24, 2025, from Classification on Web Application Requests:https://www.researchgate.net/profile/Shatha-Melhem/publication/340973761_Classification_on_Web_Application_Requests/links/

64a8c33ab9ed6874a504554d/Classification-on-Web-Application-Requests.pdf

Guillaume, H. (2012). *Failles de sécurité des applications Web : Principes, parades et bonnes pratiques de développement.* HAL. Récupéré sur https://hal.science/hal-00736013

Hamour, M., & Benhamdine, N. (2020). Prédiction du Churn Rate Par le Machine Learning dans le secteur des M&A Application au sein de KPMG(Mémoire,École Nationale polytechnique). 175. Récupéré sur https://repository.enp.edu.dz/jspui/bitstream/123456789/9814/1/HAMOUR.Mohamed Aimed_BENHAMDINE.NazimMalik.pdf

Hossen, K. (2014). Inférence automatique de modèles d'applications Web et protocoles pour la détection de vulnérabilités (Thèse de doctorat,Université de Grenoble). 172. Récupéré sur https://theses.hal.science/tel-01547286v1

Jérôme , T., & Jérôme , H. (2017). *Sécurité informatique sur le Web - Apprenez à sécuriser vos applications (management, cybersécurité, développement et opérationnel).* (E. Editions, Éd.) Récupéré sur /securite-informatique-sur-le-web-apprenez-a-securiser-vos-applications-management-cybersecurite-developpement-et- operationnel

L'équipe de relations publiques de TuxCare. (2023, Mars 14). *Attaque par exécution de code à distance : de quoi s'agit-il, comment protéger vos systèmes ?* Consulté le Mai 23, 2025, sur TuxCare: https://tuxcare.com/fr/blog/remote-code-execution-attack/

Liu, H., & Lang, B. (2019, October 17). Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. *Applied Sciences*, 28. Récupéré sur https://www.researchgate.net/publication/336630211_Machine_Learning_and_Deep_Learning_Methods_for_Intrusion_Detection_Systems_A_Survey

Mammeri, K. (2024, juin). Détection et classification des attaques DDoS dans les réseaux SDN par apprentissage machine (Mémoire de master,Université de Québec). 83.

Manipulator-in-the-middle attack. (n.d.). Retrieved 05 16, 2025, from https://owasp.org/www-community/attacks/Manipulator-in-the-middle_attack

MEGHZILI, S. (2023). *Dévloppement web avancé.* MILA, Centre universitaire de Mila. Consulté le Juin 29, 2025, sur https://elearning.centre-univ-mila.dz/a2025/pluginfile.php/140929/mod_resource/content/0/Chapitre%2003_V2-DWA.pdf

Morzeux. (2020). HttpParams. *GitHub*. Récupéré sur https://github.com/Morzeux/HttpParamsDataset

Moustafa, N., & Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion

detection systems (UNSW-NB15 network data set). In IEEE. (Ed.), *2015 Military Communications and Information Systems Conference (MilCIS)*. Retrieved Mai 24, 2025, from https://www.researchgate.net/publication/287330529

OWASP Foundation. (n.d.). *Manipulator-in-the-middle attack*. Retrieved mai 23, 2025, from OWASP: https://owasp.org/www-community/attacks/Manipulator-in-the-middle_attack

OWASP Foundation. (n.d.). *Path traversal*. Retrieved Mai 23, 2025, from OWASP: https://owasp.org/www-community/attacks/Path_Traversal

RAZAFINDRAKOTO , A. (2023, Octobre 18). Les malwares silencieux : Étude des attaques malveillantes résidents en mémoire (Mémoire de master,Université d'antananarivo). 95.

Sadqi, Y., & Chakir, O. (2024, Décembre). Demystifying the Role of Publicly Available Up-to-Date Benchmark Intrusion Datasets: A Case Study of Web Security. 20. Retrieved Mai 24, 2025, from https://www.researchgate.net/publication/389125563

Saikat Biswas, M. K. (2018). A study on remote code execution vulnerability in web applications. Retrieved from https://www.researchgate.net/publication/328956499_A_Study_on_Remote_Code_Execution_Vulnerability_in_Web_Applications

Senyuz, I., Eroglu, T., & Yildiz, O. (2023). File Inclusion. Retrieved 05 17, 2025, from https://www.scribd.com/document/789910057/Files-Inclusion-Attack

Shaheed, A., & Kurdy, M. (2022). Web Application Firewall Using Machine Learning and Features Engineering. *Security and Communication Networks*, 14. Retrieved Mai 24, 2025, from https://doi.org/10.1155/2022/5280158

Shaheed, A., & Kurdy, M. H. (2022). Web Application Firewall Using Machine Learning and Features Engineering. *Security and Communication Networks*. Syrian Virtual University, Damascus, Syria. Retrieved from https://www.researchgate.net/publication/361143640_Web_Application_Firewall_Using_Machine_Learning_and_Features_Engineering

Simon , B. (2011, Juin 6). Forêts Aléatoires: De l'Analyse des Mécanismes de(thèse de doctorat,Université de Rouen). 193. Récupéré sur https://theses.hal.science/tel-00598441v1

Sureda Riera et al. (2022). A new multi-label dataset for Web attacks CAPEC classification using machine learning techniques. *Computers and security, 120*. doi:https://doi.org/10.1016/j.cose.2022.102788

Sureda Riera et al, T. (2022). A new multi-label dataset for Web attacks CAPEC classification using. *Computers & Security*, 18.

Tavallaee, M., Bagheri, E., Ghorbani, A.-A., & Lu, W. (2009). A detailed analysis of the KDD CUP 99 data set. In IEEE (Ed.), *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications.* Ottawa, ON, Canada. Retrieved Mai 24, 2025, from https://www.researchgate.net/publication/48446353

Yassin, M. (2014, Février). Protection automatique des applications web contre l'attaque par injection SQL (Mémoire de Master, Université de Québec à Montréal). 114.