REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEINGENEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE SAAD DAHLEB - BLIDA 1 -

FACULTE DES SCIENCES

DEPARTMENT D'INFORMATIQUE

Mémoire de fin d'études

Présenté en vue de l'obtention du diplôme de master

Domaine : Mathématique & Informatique

Filière : Informatique

Spécialité : Sécurité des systèmes d'information

Thème : MISE EN PLACE D'UN MODELE DE POLITIQUE DE SÉCURITÉ DURANT LE CYCLE DE DEVELOPPEMENT ET LA MISE EN PRODUCTION DES APPLICATIONS WEB

Présenté par :

- HAMIDA ramzi
- TAIEB SOLIMANE akram

Président du jury : S. Ferfera

Promotrice : Mme REZOUG Nachida

Organisme d'accueil : université Saad Dahleb – blida 1

SESSION : octobre 2017

Résumé

La sécurité des applications web est devenue un enjeu stratégique aussi important que les fonctionnalités ou l'ergonomie, aujourd'hui, la plupart des vulnérabilités ciblent les applications web et sont donc devenus la porte d'entrée des pirates au sein d'une organisation. L'objectif de ce projet est de mettre en place un modèle de sécurité pour la phase de développement et la production des applications web. Le travail portera sur une étude générale sur les méthodes de sécurité dans les applications web et une proposition d'un modèle de sécurité qui doit être basé sur les bonnes pratiques, les technologies et les standards de la sécurité. Le renforcement du niveau de sécurité des applications web avec le pare-feu web est indispensable pour répondre aux problématiques du mauvais développement. Dans ce contexte, nous prenons part à promouvoir cette couche de protection.

Mots clés : Pare-feu web, sécurité d'information, applications web, OWASP, politique de sécurité, défense en profondeur.

ملخص

أصبح أمن تطبيقات الويب قضية استراتيجية مهمة مثلها مثل الوظائف أو بيئة العمل، واليوم، فإن معظم نقاط الضعف الأمني تستهدف تطبيقات الويب وقد أصبحت بوابة دخول القراصنة إلى اي منظمة .الهدف من هذا المشروع هو وضع نموذج أمني لمرحلة التطوير وإنتاج تطبيقات الويب .وسيركز على دراسة الطرق العامة لأمن تطبيقات الويب واقتراح نموذج أمني ينبغي أن يستند إلى أفضل الممارسات والتكنولوجيات والمعايير الأمنية .إن تعزيز المستوى الأمني لتطبيقات الويب مع جدار الحماية على شبكة الإنترنت أمر ضروري لمعالجة مشكل البرمجة غير الآمنة .وفي هذا السياق، نشارك في تعزيز وتحسين نطاق هذه الطبقة من الحماية.

الكلمات الرئيسية: جدار الحماية التطبيقي، الامن المعلوماتي، تطبيقات الويب، أواسب، السياسة الأمنية، والدفاع المعلوماتي في العمق.

Abstract

Web applications security has become a strategic issue as important as the functionalities or the ergonomics, today, most of vulnerabilities targets web applications and have become the pirate's entry gateway into an organization. The objective of this project is to set up a security model for the development phase and the production of web applications. It will focus on a study of general web application security methods and a proposal of security model that should be based on best practices, technologies and security standards. Strengthening the security level of web applications with the web firewall is essential to address the issues of poor development. In this context, we are taking part in expanding this layer of protection.

Keywords : web application firewall, information security, OWASP, security policy, defense in depth.

REMERCIEMENTS

Premièrement nous remercions Dieu de nous avoir donné foi, force et santé. Ensuite, nous remercions en particulier nos familles qui n'ont pas cessés de nous soutenir tout le long de nos études.

Nous tenons à exprimer un remerciement spéciale, et notre profonde gratitude à notre promotrice, Mme Rezoug.N docteur à l'université de Blida, pour nous avoir accordé l'honneur de travailler sous sa responsabilité, et pour tous ses conseils qui nous on éclairés tout au long de la réalisation de ce travail, elle a fait preuve d'une grande disponibilité pour l'aboutissement de ce mémoire.

Nous remercions aussi M Messaoudi Mounir pour son soutien qui nous a permis de découvrir un peu plus le monde de la sécurité informatique.

Nous remercions très sincèrement l'ensemble des membres du jury d'avoir acceptés l'invitation pour évaluer ce travail de mémoire.
Nous remercions tous les enseignants de la faculté des sciences et surtout nos enseignants du département informatique.

Nous remercions toutes les personnes qui nous ont aidées par leur soutien et leur encouragement à accomplir ce travail.

Nous remercions et dédions ce travail à nos familles et tous nos amis pour leur soutien et leur présence.

Sommaire

Liste des figures

Liste des tableaux

Liste des abréviations

ACL	Acces list control
AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
ASM	Application Security Manager
BDD	Base de donne
CAPTCHA	Completely Automated Public Turing test to Tell Computers and Humans Apart
CAPEC	Common Attack Pattern Enumeration and Classification
CLUSIF	Club de la sécurité de l'information français
CRS	Core Rule Set
CSRF	Cross-Site Request Forgery
CWE	Common Weakness Enumeratio
DICP	Disponibilité, Intégrité, Confidentialité, Preuve
DLP	Data Loss Prevention
DNS	Domain Name System
DOS	Denial of Service
DDOS	Distributed Denial of Service
ELK	Elasticsearch Logstash Kibana
ERP	enterprise resource planning
FISMA	Federal Information Security Management Act
FTP	File Transfer Protocol
HIPPA	Health Insurance Portability and Accountability Act
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secured
IDS	Intrusion Detection System
IHM	Interface Homme-Machine
IPS	Intrusion Prevention System

ISAPI	Internet Server Application Programming Interface
JAAS	Java Authentication and Authorization Security
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MS-IIS	microsoft Internet Information Services
NSM	Network and System management
OS	OPERATING SYSTEM
OSI	Open Systems Interconnection
OWASP	Open Web Application Security Project
PCI-DSS	Payment Card Industry Data Security Standard
RADIUS	Remote Authentication Dial-In User Service
RPC	remote procedure call
SANS	SysAdmin, Audit, Network, Security
SFTP	Secure Shell File Transfer Protocol
SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
Sql	Structured Query Language
SSI	Sécurité des Systèmes d'Information
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	transport layer security
URL	Uniform Resource Locator
VM	virtual machine
VPN	Virtual Private Network
WAF	Web Application Firewall
WAFEC	Web Application Firewall Evaluation Criteria
WASC	Web Application Security Consortium
WS-I	Wireless System Integration
XML	eXtensible Markup Language
XPath	eXtensible Markup Language Path
XQuery	eXtensible Markup Language Query
XSS	Cross site scripting

Introduction générale

De nos jours les applications web sont devenues omniprésentes dans tous les domaines de la vie, que ce soit dans la gestion d'une banque, une boutique, un portail client, un partenaire ou un employé. Toutes ces applications sont disponibles en permanence à tous les utilisateurs quels qu'ils soient en raison de la nature du réseau d'internet, c'est pour ça les applications Web continuent d'être une des principales cibles d'attaque pour les criminels, et la tendance ne montre aucun signe de ralentissement. En effet aujourd'hui, les attaques sont de moins en moins réalisées sur les serveurs ou sur les réseaux car ceux-ci sont de mieux en mieux protégés ,d'où le fait que la plupart des vulnérabilités se sont déplacées vers les applications web et sont donc devenus la porte d'entrée des pirates, qui à leur tour évitent les attaques réseau au profit d'attaques par cross-site Scripting *(XSS)*, d'injections SQL et de nombreuses autres techniques d'infiltration destinées à frapper plus haut dans le modèle OSI, au niveau de la couche applicative en général, et du web en particulier.

La question fondamentale est que le Web et le protocole HTTP n'ont pas étaient conçus pour des applications aussi complexes que celles qui sont actuellement à la pointe de la technologie, pour cette raison, les systèmes de sécurité informatique traditionnels tels que les pare-feu ou les systèmes de protection d'intrusion sont incapables de se prémunir totalement contre ces attaques et d'offrir une protection complète. Donc Il est très important d'utiliser des techniques complémentaires entre la sécurité des applications et la sécurité du réseau pour protéger efficacement les actifs informatiques modernes.

La sécurité dans le web est un vaste domaine qui suit une multitude d'approches qui gèrent en profondeur la sécurité de ses composants. Cependant, il est préférable de diversifier les défenses en mixant plusieurs solutions, ou approches pour augmenter le niveau de sécurité, on appelle ça la sécurité multicouche, dite « défense en profondeur ». Il est donc primordial de créer des applications Web dans une logique de sécurité dès le départ et tout au long du processus de développement et de mise en production, ainsi que de diversifier au mieux les lignes de défense de cette technologie très exposé afin d'atteindre un niveau de sécurité respectable.

II. Problématique :

Le cycle de développement typique d'une application web oublie bien souvent de prendre en considération la sécurité dès la conception. Encore trop peu de développeurs sont sensibilisés à la sécurité et il n'est pas rare qu'ils utilisent de nombreuses APIs ou des bouts de code trouvés sur le net sans en tester la robustesse. C'est pour cela que dans le domaine de la sécurité web, le cycle de développement de l'application présente toujours un souci certain, ou plusieurs problèmes se posent comme :

- Comment renforcer la protection de son application web, tout en prenant en considération les problèmes liés à la non connaissance des bonnes pratiques de codage.
- la prévention des faiblesses dès le départ et mettre en place les contrôles adaptés pour un niveau de sécurité.
- En programmant défensivement le développeur tente toujours de palier aux faiblesses de l'application web, mais malheureusement cela est toujours long et complexe car la présence des applications tierces comme une base de données au sein de l'application fait que le développeur n'a pas toujours accès à l'intégralité du code.

III. Objectif

Pour faire face à ces problèmes majeurs, il est indispensable de combiner différente mesures et couches de sécurité qui sont capables indépendamment de prendre des décisions dans certains cas d'attaques.

L'objectif de ce travail est d'étudier les différentes solutions de sécurité, et d'adopter divers moyens pour réduire le risque d'intrusion et d'altération de données sensibles.

Ci-dessous les objectifs de ce travail :

- Enumérer les différentes attaques et vulnérabilités sur le web afin de les identifier.
- Expliquer les différentes solutions existantes pour mettre en place plusieurs lignes de défense complémentaires.
- Présenter un ensemble de bonnes pratiques de programmation défensive pour la phase de développement et la production des applications Web.
- Mettre en place un guide de politique de sécurité des applications web.

- Mettre en place des règles de filtrage spécifiques à notre besoin par le biais d'un pare-feu applicatif pour une protection supplémentaire sans modifier le code.
- Rajouter un module de monitoring et d'administration pour le déploiement de notre couche de protection (pare-feu web), afin d'améliorer la politique de sécurité.

Pour arriver à ces objectifs il est essentiel de mettre en place un modèle de sécurité, où plusieurs solutions sont déployées sous la gouvernance d'une politique de sécurité fiable et efficace.

IV. Organisation du mémoire

Ce mémoire sera organisé comme suit :

Le premier chapitre comporte une introduction à la sécurité des applications web, quelques notions liés au fonctionnement du web ainsi que les failles de sécurité les plus critiques.

Le deuxième chapitre présente les différents mécanismes de protection : le codage sécurisé, les IPS, les Framework de sécurité. Puis une partie est consacrée à la présentation du WAF « le pare-feu d'application web ».

Le troisième chapitre comprend un guide d'une politique de sécurité des applications web, qui est un ensemble de recommandations et de bonnes pratiques liées à toutes les phases de développement.

Le quatrième chapitre consiste en la présentation de l'architecture d'un système de protection, nous présentons quelques parades pour atténué les attaques, et nous présentons la conception d'un module d'administration et de monitoring du pare-feu déployé, ce qui nous offre une connaissance approfondie sur le trafic entrant, et ainsi améliorer et adapter la politique de sécurité à travers le temps.

Le cinquième chapitre est consacré à la réalisation d'un module de monitoring en temps réel pour la surveillance de l'application web en cas d'attaque. Nous présentons les interfaces et une expérimentation sur un banc d'essai du prototype.

Chapitre 1

INTRODUCTION À LA SÉCURITÉ DES APPLICATIONS WEB

1. Introduction

La Sécurité des Systèmes d'Information - SSI - est une activité managériale qui consiste en un ensemble de moyens techniques, organisationnels, juridiques et humains nécessaires pour conserver, rétablir et garantir le fonctionnement normal d'un système d'information - SI - en assurant que les ressources matérielles et logicielles soient utilisées uniquement dans le cadre où il est prévu qu'elles le soient. Quand on parle de la sécurité des systèmes d'information, on parle d'un certain nombre d'objectifs ou propriétés qui doivent être vérifiés à tout moment. Selon la définition, on peut en distinguer : la confidentialité des données, leur intégrité, leur disponibilité pour les personnes qui en ont le droit, l'authenticité des utilisateurs des données et autres ressources du SI et la non-répudiation des actions de la part des prétendues personnes qui les ont effectuées [2].

Toute entreprise ou administration se doit maintenant d'avoir une présence sur le web que ce soit via un blog intranet / extranet, ces applications dîtes légères, pour leur gros avantage elles ne nécessitent qu'un navigateur web pour fonctionner et sont donc accessibles en permanence à tous public. C'est pourquoi elles sont particulièrement exposées aux attaques des pirates. Ces personnes tentent de contourner les mesures de sécurité mises en place par défis technique et intellectuel, revendication ou tout simplement pour en tirer un profit financier [3].

Dans ce chapitre, on verra les différentes terminologies qui concernent la sécurité des applications web, les risques et les vulnérabilités sur ces applications qui peuvent être importantes pour assurer que la politique de sécurité d'un SI vérifie bien les propriétés d'intégrité ,de confidentialité , de disponibilité et d'authenticité des données .

2. Le Web :

Le Web est un système qui permet de consulter à travers un navigateur des pages accessibles sur des sites. Fréquemment confondu avec Internet le Web n'est qu'une partie d'Internet, d'autres applications comme le courrier électronique, la messagerie instantanée, et le partage de fichiers en pair à pair sont d'autres parties d'internet [4].

Des applications Web ont été créées pour effectuer pratiquement toutes les fonctions utiles que vous pourriez implémenter en ligne. Voici quelques fonctions d'application Web qui ont pris de l'importance ces dernières années: Shopping (Amazon), Réseaux sociaux (Facebook), Banque (Citibank), Recherche sur le Web (Google), Enchères (eBay), jeux d'argent « gambling », Informations interactives [5].

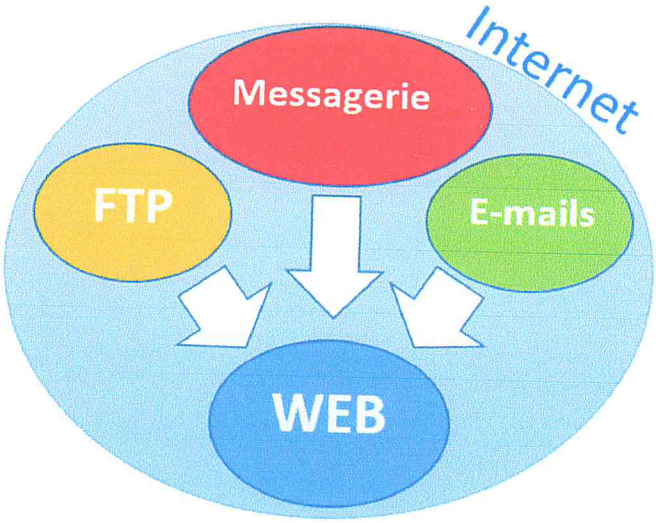


Figure 1 – représentation du web par rapport aux autres services [6].

2.1 Le Protocole http

HTTP (HyperText Transfer Protocol) est le protocole de communication de base utilisé pour accéder au World Wide Web, il est utilisé par toutes les applications Web d'aujourd'hui. C'est un modèle basé sur message ou le client envoie un message de demande et le serveur renvoie un message de réponse. Son rôle est de livrer des documents sur le Web ainsi que de permettre le transfert de fichiers localisés grâce à une chaîne de caractères nommées URL entre un client et un serveur [7].

HTTP fonctionne au niveau de la couche applicative du modèle OSI et repose entre autre sur un autre protocole : TCP (protocole réseau) Il utilise les fonctions de ce protocole de transport pour offrir un transfert fiable, ce qui permet à HTTP de se concentrer sur des aspects d'échange et de négociation des types de données échangées. L'une des forces de HTTP est sa simplicité. Son ensemble de commandes et de réponses est réduit, mais accompagné de divers en-têtes qui permettent aux clients et aux serveurs d'échanger des informations supplémentaires sur les données. Le navigateur est le client de HTTP. Il émet des requêtes à un serveur HTTP qui lui envoie les réponses. Chaque requête contient une commande spécifique qui peut être suivie de l'adresse du document suivie d'en-têtes

optionnels. La réponse du serveur inclut également des en-têtes suivis des données. Le navigateur utilise le port 80 pour communiquer avec le serveur à travers un port TCP [8].

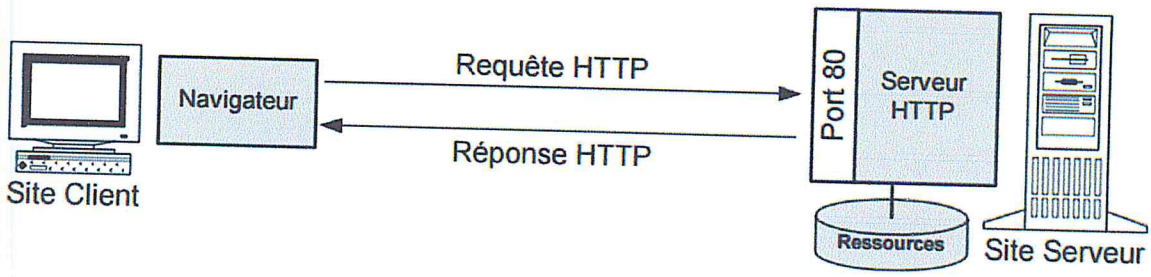


Figure 2 - le modèle d'interaction de http [8].

2.1.1 Requêtes HTTP

Tous les messages HTTP (demandes et réponses) se composent d'un ou plusieurs en-têtes, chacune sur une ligne distincte, suivie d'une ligne vierge obligatoire, suivie d'un corps de message facultatif. Un aperçue d'une requête http est dans la figure suivante *[9]:*

```
GET /auth/488/YourDetails.ashx?uid=129 HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml,
image/gif, image/pjpeg, application/x-ms-xbap, application/x-shockwave
flash, */*
Referer: https://mdsec.net/auth/488/Home.ashx
Accept-Language: en-GB
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; .NET4.0C; InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322)
Accept-Encoding: gzip, deflate
Host: mdsec.net
Connection: Keep-Alive
Cookie: SessionId=5B70C71F3FD4968935CDB6682E545476
```

Figure 3 – aperçue d'une requête http [9].

La première ligne de chaque requête HTTP se compose de trois éléments, séparés par des espaces:

1. Un verbe indiquant **la méthode HTTP**. La méthode la plus couramment utilisée GET, dont la fonction est de récupérer une ressource à partir du serveur Web. Les requêtes GET n'ont pas de corps de message, donc aucune autre donnée ne suit l'espace vide après les en-têtes des messages.
2. **L'URL demandée** c'est le nom pour la ressource étant demandé avec ces des paramètres que le client passe à cette ressource, L'exemple contient un seul paramètre avec le nom id et la valeur 129.

3. **La version HTTP** utilisée. Les seules versions HTTP courante sur internet sont 1.0 et 1.1, et la plupart des navigateurs utilisent la version 1.1 par défaut.

Voici quelques autres points d'intérêt dans un échantillon de requête :

- **Referer header** L'en-tête de référence est utilisé pour indiquer l'URL à partir de laquelle la demande Est originaire.
- **L'en-tête User-Agent** est utilisé pour fournir des informations sur le navigateur Ou un autre logiciel client qui a généré la demande.
- **L'en-tête hôte** spécifie le nom d'hôte qui apparaît dans l'URL complète à laquelle vous accédez, ceci est nécessaire lorsque plusieurs sites Web sont hébergés sur le serveur.
- **L'en-tête Cookie** permet de soumettre des paramètres supplémentaires que le serveur serveur a émis au client.

2.1.2 Réponses HTTP

Une réponse HTTP typique est la suivante [10] :

```
HTTP/1.1 200 OK
Date: Tue, 19 Apr 2011 09:23:32 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Set-Cookie:  tracking=tI8rk7joMx44S2Uu85nSWc
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1067
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://
www.w3.org/1999/xhtml" ><head><title>Your details</title>
...
```

Figure 4 - aperçue d'une réponse http [10].

La première ligne de chaque réponse HTTP se compose de trois éléments, séparés par des espaces :

1. La **version HTTP** utilisée.
2. Un numéro de **code d'état** indiquant le résultat de la demande, par exemple 200 est le code d'état qui signifie que la demande est réussie et que la ressource est retournée.
3. Une «phrase de motif» textuelle décrivant davantage l'état de la réponse.

Voici quelques autres points dans la réponse se compose:

- **L'en-tête du serveur** contient une bannière indiquant le logiciel du serveur Web.
- **L'en-tête Set-Cookie** donne le navigateur un cookie, Ceci est renvoyé dans l'en-tête Cookie des demandes ultérieures à ce serveur.
- **L'en-tête Pragma** indique au navigateur de ne pas enregistrer la réponse dans son cache.
- **L'en-tête Content-Type** indique que le corps de ce message contient un document HTML.
- **L'en-tête Content-Length** indique la longueur du corps du message en octets.

2.1.3 Les Méthodes HTTP

HTTP définit un ensemble de méthodes de requête qui indiquent l'action que l'on souhaite réaliser sur la ressource indiquée. Chacune implémente une sémantique différente mais certaines fonctionnalités courantes peuvent être partagées par différentes méthodes (e.g. une méthode de requête peut être sûre, idempotente ou être mise en cache) [11].

***Tableau 1** : les différentes méthodes http [11].*

Méthode http	Description
GET	Extrait une ressource.
POST	Crée une ressource.
PUT	Met à jour une ressource existante, mais ne la crée pas si elle n'existe pas.
DELETE	Supprime une ressource.
HEAD	Même que GET mais retourne seulement en-têtes HTTP et aucun corps du document
TRACE	La méthode TRACE réalise un message de test aller/retour en suivant le chemin.

OPTION	La méthode OPTIONS est utilisée pour décrire les options de communications avec LE SERVEUR.

2.1.4 Les code de status http

Les codes statut de réponse font référence à des requêtes internet faites par des moteurs de recherche ou des visiteurs. Ces codes sont toujours représentés par trois chiffres commençant par 1.2.3.4 ou 5. Ce code indique le statut d'un élément internet [12].

De 100 à 500, les status codes sont divisés selon les catégories suivantes :

100 : informatif. La requête a bien été reçue et le processus est en marche.
200 : succès. La requête a bien été reçue et marche correctement.
300 : redirection. La requête a été reçue mais une étape supplémentaire est nécessaire pour qu'elle soit complétée.
400 : erreur client : la requête a été demande par un client mais la page de destination est incorrecte.
500 : erreur serveur : la requête fournie par le client est correcte mais le serveur a échoué dans la transmission de sa réponse.

Figure 5 – les codes d'état http [12].

Il existe de nombreux codes d'état spécifiques qui sont utilisés uniquement dans des circonstances spécialisées.

2.1.5 HTTPS

Le protocole HTTP utilise TCP simple comme son mécanisme de transport, qui n'est pas chiffré et peut donc être intercepté par un attaquant qui est bien positionné sur le réseau. HTTPS est essentiellement le même protocole de couche d'application comme HTTP mais luis il est tunelé sur un mécanisme de transport sécurisé, Secure Sockets Layer (SSL). Cela protège la vie privée et l'intégrité des données sur le réseau, ce qui réduit les possibilités d'interceptions. Les requêtes et les réponses http fonctionnent exactement de la même manière, sans prendre en considération le protocole utilisé pour le transport (SSL) [13].

NOTE : SSL a été strictement remplacé par transport layer security (TLS), mais par abus de langage ce dernier est encore appelé par l'ancien nom SSL.

3. Les critères fondamentaux de la sécurité de l'information

Afin de disposer d'une infrastructure sécurisée les critères DICP doivent être respectés. Pour rappel, voici la définition de ces critères [14] :

- **Disponibilité** : l'information doit être continuellement disponible.
- **Intégrité** : l'information ne doit pas être altérée.
- **Confidentialité** : l'information doit être uniquement accessible aux personnes autorisées.
- **Preuve/Traçabilité** : Tous les mouvements de données doivent être tracés.

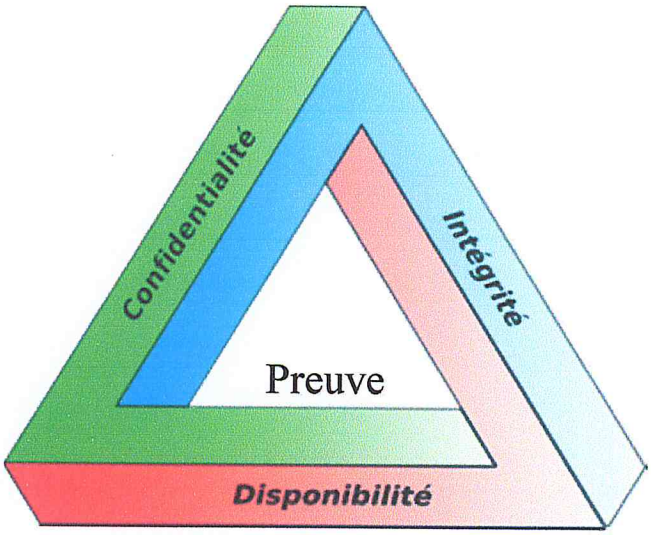


Figure 6 - Critères D I C P [41]

- Chaque application dispose de ses propres critères DICP en fonction de la politique de sécurité de l'entreprise ainsi que des besoins d'affaires exprimés par le client. Les besoins en disponibilité d'une application ou la confidentialité des données peuvent être plus ou moins forts.

4. Notions sur la sécurité des applications web

La sécurité est un investissement et une chaîne où chaque composant et chaque acteur apportent des risques, et pour s'engager à déployer une politique de sécurité il faut mesurer certain aspect pour pouvoir y est procéder :

Menace : "Une violation probable de la sécurité", Une menace se présente sous de multiples formes : il peut s'agit d'un virus ou d'un vers informatique, d'une personne sur Internet, d'un mail de phishing ou encore d'un intrus tentant de rentrer dans un local sécurisé. Assez souvent, la menace est aussi appelée "attaquant". Il faut cependant rester large dans la définition d'une menace car l'eau, la température ou l'humidité sont aussi des menaces. Tout ne se résume donc pas à des pirates informatiques ou à des virus, il s'agit en générale de l'attaquant ! [16].

Vulnérabilité : Une vulnérabilité peut aussi être appelée "faille" .
C'est une faiblesse qui peut être utilisée par une partie pour amener le logiciel à manipuler des données non désirées, interrompre l'exécution correcte ou effectuer des actions qui n'étaient pas spécifiquement accordées à la partie qui exploite la faiblesse. Les vulnérabilités peuvent être techniques ou humaines, un système qui n'est pas à jour des correctifs de sécurité, une climatisation défectueuse ou des personnes utilisant des mots de passe trop simples sont des exemples de vulnérabilités [15] [16].

Impacte : C'est les conséquences d'une attaque réalisé ou la faiblesse est présente, simplement, c'est les dégâts pour une organisation ou un environnement En cas d'attaque [15].

Attaque : Un ensemble d'actions et de tentative bien définie cherchant endommager le fonctionnement d'un système [16].

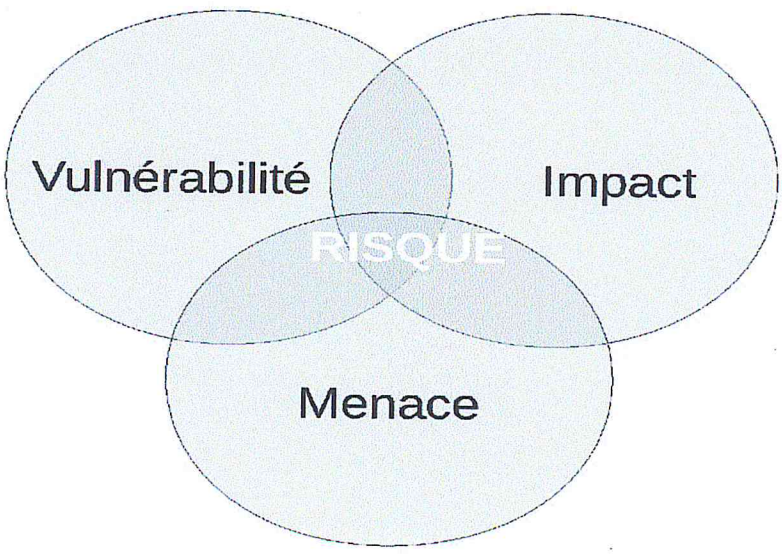


*Figure 7 - composants d'un risque en SSI / Risque = impact * vulnérabilité * menace [42]*

5. Cibles des attaques Web

Johanne Ulloa *[17]* résume les éléments constituant l'interaction des Web App :

- **L'utilisateur :** essentiellement, les attaques visant l'utilisateur cherchent à récupérer des éléments d'authentification afin d'usurper son identité et réaliser toutes les actions que ce dernier aurait pu faire.
- **Le serveur Web :** occupe une importance stratégique et les impacts les plus fréquents sont:
  - La modification de contenu : les conséquences du d'effacement sont variables en fonction de la sensibilité de l'image de l'organisme ;
  - L'ajout de contenu illicite ;
  - L'accès à des données confidentielles présentes ;
  - La prise de contrôle du serveur Web (exécution de commandes), le cas des zombis.
- **L'application Web** : dans ce cas, le but sera de faire en sorte que l'application Web ait un comportement différent de celui attendu. Par exemple : modifier le prix d'un produit.
- **Les bases de données :** elles sont la cible privilégiée des hackers. Les impacts vont du vol, à la suppression ou l'altération de données. Notons que bien des crimes passent complètement inaperçus, du fait que la victime ne s'en rend pas compte de l'effraction, l'exemple du vol.
- **Les services Web :** n'échappent pas aux attaques. De plus en plus déployés, ils ont une forte interaction avec les bases de données, ce qui fait d'eux une cible de choix.

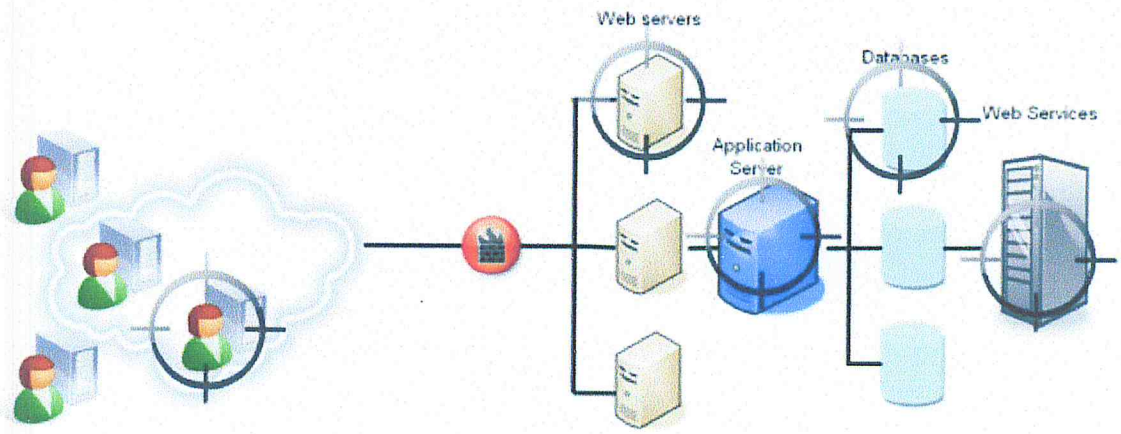


Figure 8 - Cibles des attaques Web [17]

6. Les vulnérabilités des applications web les plus critiques

De nombreuses études publiques, comme celles des communautés OWASP, SANS WASC ou CLUSIF [annexe], nous révèlent que la grande majorité des vulnérabilités sont d'ordres applicatifs.

Avoir connaissance de ses vulnérabilités est un élément clef pour mettre en place une stratégie de sécurité. Les vulnérabilités techniques sont typiquement identifiées lors d'audits de sécurité ou via des outils (scanners de ports ou de vulnérabilités).

Ci-dessous un classement du top 10 des vulnérabilités les plus critiques à qui il faut se méfier [18] :

Risques classé par l'OWASP	Attaques classé par le WASC	Catégories
Injection	SQL Injection (WASC-19) XML Injection (WASC-23) Null Byte Injection (WASC-28) LDAP Injection (WASC-29) Mail Command Injection (WASC-30) OS Commanding (WASC-31) XPath Injection (WASC-39) XQuery Injection (WASC-46)	Exécution de commandes
Violation de Gestion d'authentification et de Session	Insufficient Authentication (WASC-01) Brute Force (WASC-11) Credential/Session Prediction (W-18) Session Fixation (WASC-37) Insufficient Session Expiration (W-47)	Autorisation, Authentification
Cross-Site Scripting (XSS)	Cross-Site Scripting (WASC-08)	Attaques côté client
Références directes non sécurisées à un objet	Insufficient Authentication (WASC-01) Insufficient Authorization (WASC-02) Path Traversal (WASC-33) Predictable Location (WASC-34)	Autorisation, Authentification, Révélation d'informations
Mauvaise configuration sécurité	Server Misconfiguration (WASC-14) Application Misconfiguration (W-15)	Révélation d'informations
Exposition de données sensibles	Insufficient Data Protection (WASC-50)	Révélation d'informations

Manque de contrôle d'accès au niveau fonctionnel	Insufficient Authorization (WASC-02) Denial of Service (WASC-10) Brute Force (WASC-11) Insufficient Anti-automation (W-21) Predictable Location (WASC-34)	Autorisation, Authentification, Révélation d'informations
Falsification de requête intersites (CSRF)	Cross-Site Request Forgery (WASC-09)	Attaques logiques
Utilisation de composants avec des vulnérabilités connues	Non classe par wasc	/
Redirection et Renvois non validés	URL Redirector Abuse (WASC-38)	Attaques logiques

Tableau 2 : *les vulnérabilités web les plus critiques selon OWASP et WASC*

La figure ci-dessous représente la répartition des attaques web en pourcentage selon le rapport publié par ERPscan 2016 [43].

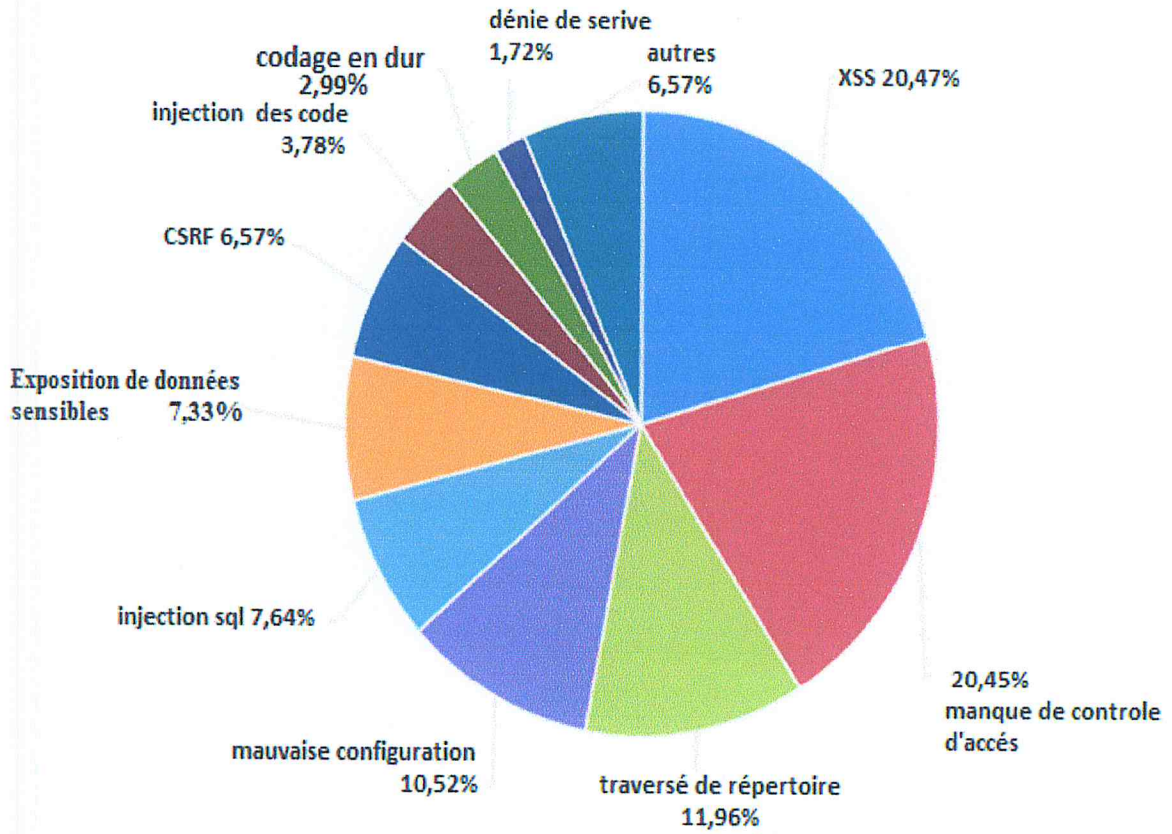


Figure 9 – Répartition d'attaques (ERPscan 2016) [43].

6.1 Évaluation des vulnérabilités

Il est évident que l'évaluation des vulnérabilités est une étape primordiale dans la validation d'un système, dans ce contexte, on trouve :

• Les audits de sécurité sont une évaluation du niveau de sécurité qui s'appuie sur un tiers de confiance comme les sociétés spécialisées en sécurité informatique afin de valider les moyens de protection mis en œuvre, au regard de la politique de sécurité [19].

• Les tests d'intrusion ou pentest sont une pratique d'audit technique. On distingue principalement trois catégories :

- La méthode dite « boîte noire » (en anglais « black box ») consiste à essayer d'infiltrer le système de l'extérieur, autrement dit, sans avoir de connaissances préalables de celui-ci, afin de réaliser un test en situation réelle ;
- La méthode dite « boîte blanche » (en anglais « white box ») consistant à tenter de s'introduire dans un système « transparent », c.-à-d., en ayant connaissance de l'ensemble du système, afin d'éprouver au maximum sa sécurité.
- Une troisième approche, dite « boite grise » où le testeur peut n'avoir qu'un nombre limité d'informations.

7. Conclusion :

Dans ce chapitre nous avons présenté une introduction à la sécurité des application web, avant tous nous avons présenté le Web et l'anatomie du protocole majeur utilisé qui est le HTTP, nous avons expliqué la forme et le principe des requêtes et réponses de ce protocole puis on s'est intéressé aux faiblesses des applications web.

Bien qu'il existe beaucoup d'autres vulnérabilités, nous avons énuméré les failles essentielles classé par les organismes les plus versé dans la sécurité web, de même on a décrit les différentes cibles ainsi que les impacts des attaques, toutes ces notions doivent être différenciées et évaluées afin d'avoir un diagnostic cohérent.

Dans le prochain chapitre nous allons étudier des solutions de sécurité web et présenter les différentes lignes de protection.

Chapitre 2

Solutions de sécurité Web

1. Introduction

Aujourd'hui la plus part d'architectures réseaux sont protégées contre des attaques venant de l'extérieur par des firewalls, IPS...etc, mais dans le plus part des cas, les application web et leurs code source sont désormais le vecteur le plus important des attaques dirigées contre la sécurité des systèmes d'information, à cet effet, nous ne pouvons plus nous permettre de tolérer les problèmes les plus simples comme ceux présentés dans le Top 10 OWASP qui sont dus principalement à un développement et un déploiement non sécurisé .

Un code source sécurisé protège bien notre application, mais en réalité il est conseillé de mettre en œuvre plusieurs mesures de protection indépendantes permettant de renforcer notre sécurité face aux menaces envisagées. Il est souvent beaucoup plus facile d'appliquer ce principe si le système à sécuriser est découpé en éléments nettement séparés et possédant chacun leurs propres mécanismes de sécurité. En effet, on appelle ça *la défense en profondeur*. De la Sécurité de l'infrastructure (matériel et logiciel) jusqu'au code source en passant par le déploiement des solutions comme le pare-feu applicative, chaque composant doit assurer sa propre protection [20].

Ce chapitre présente les différentes méthodes de protection et les solutions de sécurité qui représente les différentes barrières de protection, à côté de ça on va étudier quelques projets phares des pare-feu web, puis à la fin nous expliquons la solution qui seras exploité et amélioré de façon que cette solution sois déployer convenablement.

2. Mesures de sécurité

La protection contre les nombreuses attaques web se fait depuis plusieurs lignes de défense, cela consiste à exploiter plusieurs techniques de sécurité afin de réduire le risque lorsqu'un composant particulier de sécurité est compromis ou défaillant, cette technique est appelé la défense en profondeur, le principe revient à sécuriser chaque sous-ensemble du système où chaque composant ne fait pas confiance à la robustesse des autres couches. Ainsi, chaque composant effectue lui-même toutes les validations nécessaires pour garantir la sécurité.
Il est présenté dans ce qui suit les principales lignes de défense pour une application web:

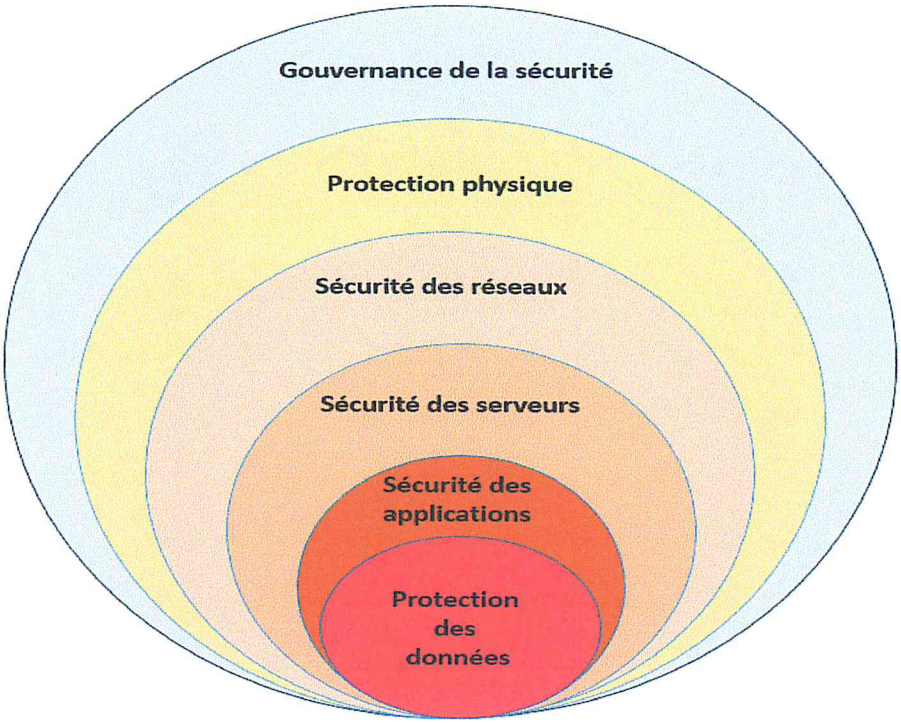


Figure 10 - illustration montrent les lignes de défense en profondeur

2.1 Un Code source sécurisé

Le meilleur endroit pour lutter contre les attaques d'applications Web est dans le code source lui-même. Les développeurs peuvent surpasser la plupart des types d'attaques en suivant les bonnes normes de codage telles que la manipulation des erreurs et la validation des entrées/sorties des données échangées avec les clients [21]. Donc, il s'agit de prendre la sécurité en tête dès le démarrage du projet, ce que l'on désigne par : « une stratégie de programmation défensive ». On peut énumérer une liste de mesures communes qui peuvent être employées par les développeurs pour améliorer la sécurité de leurs applications :

- **Code Source** : il est préférable que le code source soit clairement lisible, il faut vérifier que les commentaires sont inclus dans les délimiteurs du langage pour qu'elles n'apparaissent pas dans le source HTML reçu par le navigateur.

- **Authentification** : il s'agit de durcir les informations d'accès en les rendant plus aléatoires, elles doivent être sauvegardées sur la base de données dans un format bien crypté. Les actions sensibles des utilisateurs doivent être faites après une réauthentification. Il est également préférable que l'application informe l'utilisateur des tentatives d'accès échouées.

- **Manipulation de sessions** : le jeton de la session doit, autant que possible, être aléatoire, il est nécessaire d'en changer périodiquement pour minimiser les attaques de spoofing. L'application doit poursuivre ou bloquer les accès concurrents, cela peut bloquer les attaques de session hijacking. L'application doit terminer les sessions inactives automatiquement et achever totalement celles des utilisateurs déconnectés.

- **Manipulation des erreurs** : les erreurs doivent être contrôlées le maximum possible, les pages d'erreurs ne doivent pas fournir des informations sur l'état de l'application comme les noms de variables, les noms de fichiers ou les requêtes de base de données. Au lieu de ça, l'application devra écrire dans un fichier « error_log » spécial, ce fichier fournira les informations nécessaires pour le débogage.

- **Manipulation de la base de données :** les informations de connexion à la BDD doivent être stockées dans un endroit sécurisé. Si le nom de l'utilisateur et son mot de passe sont stockés en clair dans un fichier, il est nécessaire de vérifier que ce fichier ne peut être lu qu'à partir de l'application « permissions système - ACL ». Les requêtes SQL doivent être faites via un utilisateur qui a le minimum possible de privilèges sur la base de données. Il faut forcer les types de chacune des variables utilisées dans la requête (ex. : le champ numérique doit utiliser une variable de type entier).

- **Manipulation des fichiers :** les références vers les fichiers des « variables manipulées par les utilisateurs » doivent être nettoyées des caractères de parcours (ex. ../). Les fichiers doivent être récupérés à partir du même répertoire et ce dernier ne doit pas contenir le code source de l'application. Le répertoire qui contient les fichiers téléversés ne doit pas autoriser l'exécution.

- **Validation des entrées :** tous les caractères doivent être placés dans leurs représentations attendues. L'application doit vérifier la longueur des données et les données supplémentaires doivent être tronquées et ignorées par l'application. Le contenu invalide des données est contrôlé, l'application vérifie d'une manière proactive les « mauvais » caractères (ex. ' ; < > et les parenthèses). L'application doit valider les données restreintes et spécifiques (ex. Les Wilayas de l'Algérie sont des chaînes de caractères, mais l'entrée est seulement une des 48 possibilités, l'entrée « AdsAE » doit être rejetée par l'application).

2.1.1 Audit du code source :

Le but principal de cette méthode est de détecter les bugs et les failles de sécurité et de les corriger dans la phase de développement. Il y a deux types d'audit du code source [22]:

a) **L'audit du code manuel :**

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

b) **L'audit du code automatique :**

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

2.1.2 Frameworks de sécurité :

Afin de rendre la sécurité moins complexe pour les développeurs, l'idée de mettre en œuvre des frameworks spécialisés dans la fourniture de routines pour la sécurité est surgie. Généralement, ces fonctions simplifient et en même temps, renforce l'authentification, l'autorisation, la cryptographie et la gestion des sessions [21].

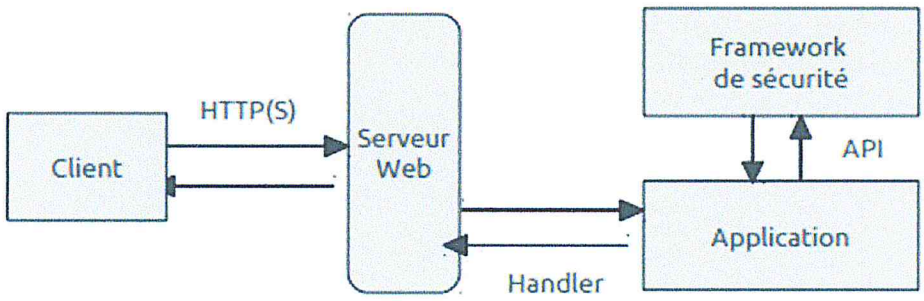


Figure 11 - Diagramme de communication général pour un framework de sécurité

Parmi les frameworks open-sources spécialisés dans la sécurité, on en trouve par exemple pour les applications écrites en Java :

- **Apache Shiro** conçu pour être un framework intuitif et facile à utiliser tout en offrant des fonctions de sécurité robustes.
- **Spring Security** est un framework Java/Java EE qui est largement utilisé pour la sécurité des applications basées sur le framework Spring.
- **jGuard** est basé sur JAAS (Java Authentication and Authorization Security) et est écrit pour les applications web ainsi que les applications autonomes, pour résoudre plus simplement, les problèmes liés au contrôle d'accès.

Aujourd'hui la majorité des frameworks de développement intègrent à leurs fonctionnalités la prise en charge de la sécurité, mais nécessitent d'avoir une maîtrise de l'API utilisée. C'est très utile pour construire des applications sur une stratégie de programmation défensive, mais que faire pour les applications déjà construites et désirant assurer plus leur sécurité.

2.2 Les IPS (Systèmes de prévention des intrusions) :

Les IPS sont des systèmes qui se trouve généralement en écoute et surveille le trafic réseau lorsque les paquets traversent. Les systèmes de prévention des intrusions (IPS) fonctionnent principalement au niveau du réseau et n'ont qu'une capacité très limitée à identifier les attaques au niveau de l'application (couche OSI 7), Surtout si il s'agit d'un trafic crypté SSL, même si certains IPS sont en mesure de le décrypter en téléchargeant la clé privée sur ces systèmes, ça reste une tâche énorme car chaque serrure SSL doit être capturée afin de pouvoir complètement décoder le trafic. Du point de vu performance et de sécurité les IPS ne peuvent identifier qu'un nombre très limité d'attaques d'application web car étant donné qu'il existe un grand nombre de technologie Web, il y aura beaucoup de différents types de vulnérabilités disponibles pour les attaquants à exploiter [23]. Les IPS ne peuvent pas les couvrir toutes efficacement ,et en réalité peuvent finir par produire plus de faux positifs que font des analystes de sécurité déjà occupés encore plus occupés Une surcharge de faux positifs peut retarder la réponse aux attaques réelles ou acceptées normalement des attaques à cause d'un analyste essayant de réduire le bruit [24].

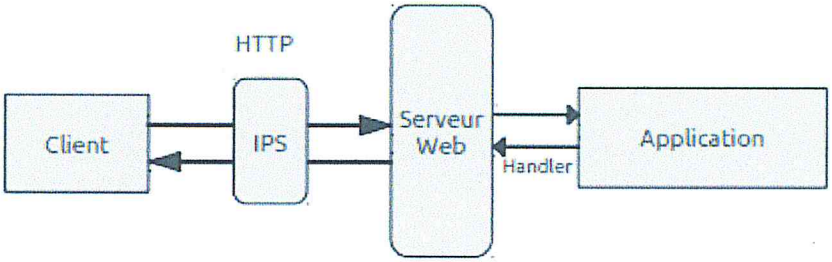


Figure 12 - l'un des déploiements possibleS d'un IPS

2.3 Les Pare-feux applicatif (WAF: Web Application Firewall)

Un pare-feu applicatif web (en anglais, Web Application Firewall ou WAF) est une partie logicielle ou un équipement matériel qui permet principalement de protéger les applications Web des attaques applicatives (SQL injections, Cross Site Scripting, injection de code…etc.), Il intercepte toutes les requêtes HTTP et analyse chacune d'entre elles avant d'accéder au serveur Web pour traitement, il analyse les requêtes GET et POST tout en appliquant des règles définies pour identifier et filtrer le trafic illégitime [25].

Selon les options WAF sélectionnées, il interroge le comportement et la logique de ce qui est demandé et renvoyé, il peut bloquer le trafic, défier le visiteur (en leur demandant de saisir un CAPTCHA ou des options de défie et de blocage), les WAFs détecte non seulement les attaques connues dans les environnements d'applications Web, mais aussi détecte (et peut prévenir) de nouveaux types inconnus d'attaques en regardant des modèles inhabituels ou inattendus dans le trafic, et il peut alerter et / ou se défendre contre ces attaques inconnues, par exemple, si un WAF détecte que l'application renvoie beaucoup plus de données que prévu, il peut la bloquer et avertir quelqu'un [24].

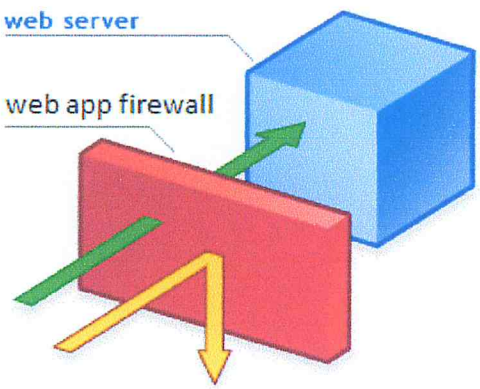


Figure 13 - Schéma de fonctionnement d'un WAF

Le WAF c'est aussi une protection immédiate sans toucher au code, grâce au Virtual Patching, c.-à-d. corrigé directement sur le WAF les vulnérabilités présentes sur l'application, car la mise en place d'un patch au sein de l'application peut-être beaucoup plus long et complexe que la création d'une règle sur le WAF. En effet l'utilisation d'applications tierces (base de données …etc.) au sein de l'application fait que le développeur n'a pas toujours la maitrise de l'intégralité code. Du coup Le niveau d'utilité et d'efficacité des WAF dépendent de certains facteurs, comme le statut de l'infrastructure, l'application, son code source ou son développement [1].

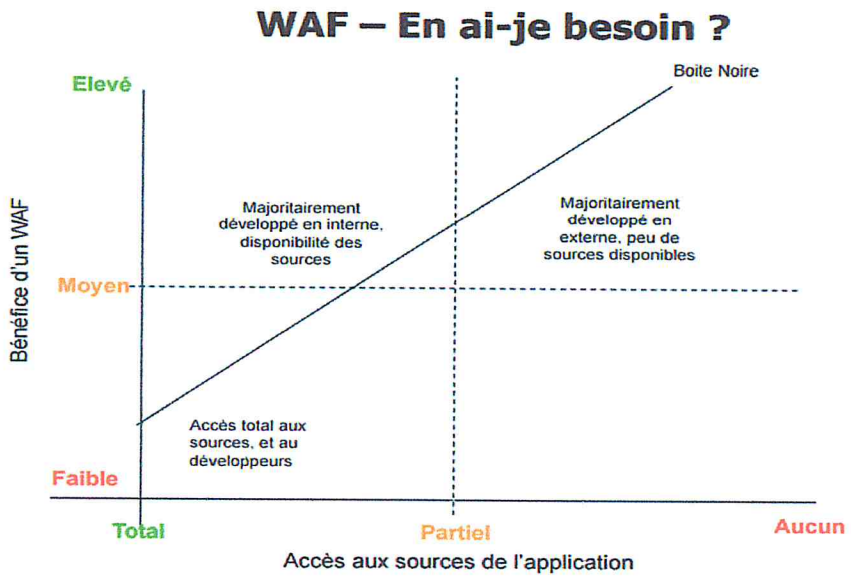


Figure 14 - Niveau d'utilité des WAF (OWASP) [45]

2.3.1 Modèles de sécurité

Le fonctionnement du WAF est généralement basé sur l'un des trois modèles de sécurité:

Figure 15 - illustration des modèles de sécurité positive et négative [44]

- **Liste noire ou modèle de sécurité négatif** - Cela utilise des signatures génériques pour protéger le site contre les attaques connues et les signatures spécifiques afin de prévenir les attaques susceptibles d'exploiter les vulnérabilités dans l'application Web.

- **Liste blanche ou modèle de sécurité positif** - Cela utilise des signatures et parfois une logique supplémentaire pour ne permettre que le trafic répondant à certains critères. Un exemple permet d'autoriser uniquement les requêtes HTTP GET à partir d'une URL spécifique et de bloquer tout le reste.
- **Modèle de sécurité hybride** - Cela s'applique à la fois aux modèles négatifs et positifs. Certaines des options configurables comprennent le blocage de la requête, le blocage de la session, le blocage de l'adresse IP, le blocage de l'utilisateur ou la connexion à l'utilisateur.

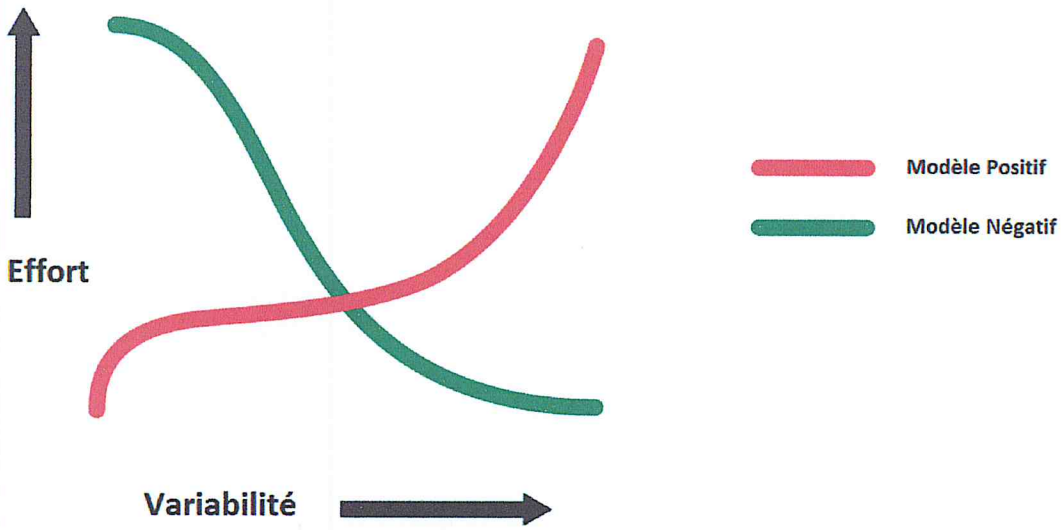


Figure 16 - L'effort nécessaire en fonction de la variabilité des applications à sécuriser [44]

***Tableau 3** - comparaison modèle négatif et modèle positif [45]*

	Négatif	Positif
Concept	Le WAF reconnait les attaques et les bloques, il autorise tous les accès.	Le WAF connait le trafic légitime et rejette tout le reste.
Avantages	• Aucun besoin de personnalisation • Protection standard • Simple à déployer	• Bloque les attaques inconnues. • N'est pas dépendant de la base de signatures. • Détection précise.
Inconvénients	• Extrêmement dépendant des signatures. • Pas très précis.	• Configuration complexe. • Sensible aux faux positifs.

2.3.2 Modes de déploiement d'un pare-feu applicatif

Il existe plusieurs modes d'opération pour les WAF, chaque mode a des avantages et des inconvénients qui nécessitent une étude pour évaluer celui approprié.

- **Reverse proxy :** le mode reverse proxy est le mode le plus commun du déploiement des WAF. Il consiste à mettre le WAF en frontal du serveur Web. Ce mode de fonctionnement présente de nombreux avantages tant en termes de sécurité qu'en termes de performances ou d'intégration dans des architectures complexes. Ce mode opératoire permet en premier lieu de masquer l'infrastructure hébergeant l'application Web. En effet, le seul point d'accès étant le reverse proxy, l'utilisateur n'a par conséquent aucune visibilité de cette infrastructure.
- **Transparent Proxy** : le mode transparent impose que le WAF soit mis en œuvre sur un lien physique supportant l'intégralité du trafic à destination des serveurs à protéger. Cela impose de laisser les équipements protégés dans des zones de sécurité accessibles depuis l'extérieur.
- **Intégré (Host Based)** : les WAF de ce type sont des applications installées sur le serveur lui-même comme plugin ou module en général.

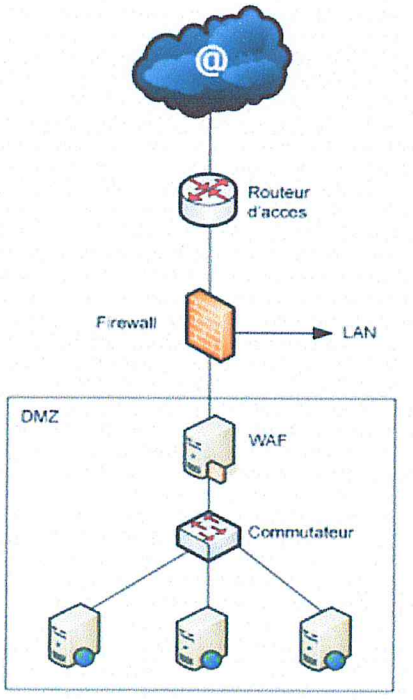


Figure 17 - Mode transparent d'un WAF.(CLUSIF) [19]

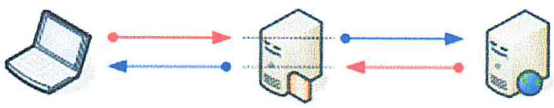


Figure 18 - Mode réverse proxy d'un WAF. (CLUSIF) [19]

- **Monitoring Mode :** le pare-feu n'est pas aligné avec le sens de la communication, mais le surveille via un port, idéal pour les tests, il peut intervenir pour interrompre le trafic indésirable en envoyant au dispositif auquel il est lié (via TCP). Il n'inflige pas de latence supplémentaire, mais il n'est pas complètement efficace du fait qu'il n'est pas un intermédiaire direct.

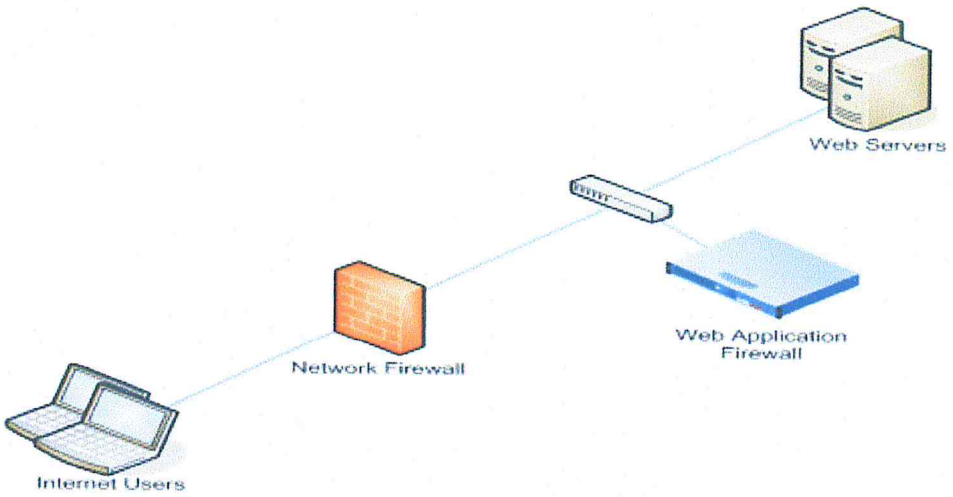


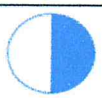
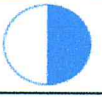
Figure 19 - WAF en mode monitoring. (impreva) [19]

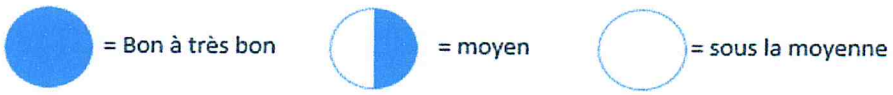
2.4. Pourquoi un WAF et non pas un IPS

Les IPS ne peuvent identifier qu'un nombre très limité d'attaques d'application web tandis que les WAF terminent et sécurisent toutes les applications Web au niveau de la couche 7 du modèle OSI, Les IPS n'ont pas la capacité de comprendre la logique du protocole d'application Web, Cette courte échéance pourrait permettre sois des attaques sans détection ni prévention, en particulier les attaques les plus récentes sans signature, ou alors peuvent finir par produire plus de faux positifs [24].

Cependant, le personnel de sécurité ne parvient souvent pas à bien expliquer comment les WAF peuvent fournir des garanties plus larges et plus détaillées des applications Web que les IPS. Le tableau suivant met en évidence les différences entre les IPS et les WAF en matière de sécurité des applications Web [23].

Tableau 4 *- explication de différence de fonctionnement entre l'IPS et le WAF* [31].

	Web Application Firewall	Intrusion Prevention System
Multi protocole sécurité		
Réputation IP		
Signature d'attaque web		
Signatures web vulnérabilités		
Apprentissage automatique des politiques		
Url, paramètre, cookies et protection de forme		
Influence sur Résultats d'analyse de vulnérabilité		



Le tableau ci-dessus montre les différenciations entre ces deux solutions dans lequel on peut apercevoir les points fort du WAF par rapport à l'IPS dans la signature d'attaques web, dans le filtrage des URL, Cookie, et paramètre http, de plus que le WAF peut-être exploiter pour automatisé une politique de sécurité. Contrairement à l'IPS qui est moyen dans cette surface (le web) alors que son avantage c'est qu'il offre une protection pour plusieurs protocole.

3. Solutions WAF existantes

Cette partie se concentrera sur les meilleures offres de produits WAF commerciale et open source et nous examinerons les caractéristiques majeures qui sont généralement associées:

3.1 Solutions commerciales : Beaucoup de WAF commerciaux ont des fonctionnalités similaires, mais des différences majeures se réfèrent souvent aux interfaces utilisateur, aux options de déploiement ou aux exigences dans des environnements spécifiques [26].

Citrix NetScaler Application Firewall :

Citrix (appFirewall) fournit une solution complète qui offre une protection contre un certain nombre de menaces communes ou inconnues. Il fournit la possibilité d'effectuer une inspection en profondeur de HTTP, HTTPS et XML ainsi que le support du top 10 d'OWASP. Sa protection contre les menaces comprend [26]:

- Les Attaques SQL Injection
- Les attaques Cross-Site Scripting
- Falsification de cookie
- Protection et validation de forme
- Validation du format de réponse et de requête HTTP et XML
- Protection par signature et par comportement
- Inspection des payload JSON
- la prévention de pertes de données (DLP), y compris la surveillance du trafic pour l'exposition intentionnelle et non intentionnelle des données
- Prise en charge de l'authentification, de l'autorisation et de l'audit
- Supporte le SSL offloading
- Protection dénie de service couches 4-7

F5 - BIG-IP Application Security Manager

F5 est un autre fournisseur bien respecté, ASM fournit une solution complète qui offre une protection contre un certain nombre de menaces communes et peu communes (ou inconnues). Il fournit la protection contre les attaques ciblées HTTP, HTTPS et XML avancées sur le support top 10 d'OWASP. Sa protection contre les menaces comprend [26]:

- Les attaques SQL injection
- Protection et validation de Forme
- Supporte le SSL offloading
- Détournement de sessions
- Protections a base de signature et de comportement
- La prévention de pertes de données (DLP), y compris la surveillance du trafic pour l'exposition intentionnelle et non intentionnelle des données
- Protection dénie de service couches 4-7, Défense contre les Botnet

- Les attaques Cross-Site Scripting
- Falsification de Cookie
- Validation du format de réponse et de requête HTTP et XML
- inspection des payload JSON
- Service F5 IP Intelligence offrant une protection contre les sources de menace connues, y compris la construction de politique en fonction des emplacements géographiques
- Authentification, y compris LDAP and RADIUS
- Outils de reporting et de politique qui permettent une verification facile de conformité PCI-DSS, HIPAA, SOX et BASEL II
- Protection supplimentaire pour les attaques SMTP and FTP (SPAM, Antivirus, Harvesting)
- Patch Virtuel

Imperva SecureSphere

La solution Imperva SecureSphere WAF existe depuis longtemps et, à cause de cela, elle est considérée comme l'une des meilleures options de pure play disponibles. La solution SecureSphere est l'une des offres WAF les plus complètes disponibles, considéré comme trop robuste pour certains déploiements de petites entreprises, et en raison de cette robustesse, il est également trop coûteux.

Cette solution support le top 10 d'OWASP. Il offre une protection pour un certain nombre de menaces, y compris (mais sans s'y limiter):

- Les attaques SQL Injection
- Détournement de Session
- Les attaques Cross-Site Scripting
- Falsification Cookie
- Site Scraping
- Patch Virtuel
- Stateful Firewall
- Validation du format de réponse et de requête HTTP et XML
- Validation du format de réponse et de requête HTTP et XML
- protection DoS/DDoS, /Botnet Protection
- SOAP, HTML5 sockets, Web 2.0 Protections

- Service ThreatRadar Reputation offrant une protection contre les sources de menace connues, y compris la construction de politique en fonction des emplacements géographiques
- Protection signature et comportement
- La prévention de pertes de données (DLP), y compris la surveillance du trafic pour l'exposition intentionnelle et non intentionnelle des donnée
- Outils de reporting et de politique qui permettent le controle de conformité PCI-DSS, HIPPA, SOS et FISMA

Le tableau suivant offre un aperçue de performance et prix sur ces trois WAFs

Produit	Transaction SSL	Débits	prix
Citrix netscaler	1500	500 mbps	4000 $
Impreva secure sphere x 2010	2200	500 mbps	4200 $
F5 BIG-IP 102000	75000	5000 mbps	84995 $

Tableau 5 *– tableau comparatif entre trois produits WAF* [26]

3.2 Solutions non commerciales open sources

Il existe des dizaines de projets WAF, mais la plupart sont conçus pour des tâches spécifiques ou pour des raisons de recherche, comme les projets de classe business, pour notre part, nous parlerons principalement de trois WAF très répandus.

ModSecurity :

Le WAF le plus utilisé est le pare-feu open source ModSecurity [27]. ModSecurity est actuellement maintenu par trustwave [28], une société qui vend également des appareils commerciaux qui contiennent cette solution. Ce WAF fonctionne comme module sur de nombreux serveurs web dont Apache, MS IIS et NginX [29].

Les règles de base de ModSecurity sont un ensemble de règles qui détectent les attaques Web les plus courantes. ModSecurity a plusieurs fonctionnalités liées au domaine de la sécurité web, tels que :

– Le support des modèles positifs et négatifs.
– Règles basées sur les expressions régulières.
– Possibilité de filtrer le corps que ce soit pour la requête ou pour la réponse.
– Structuration des données sous des variables prédéfinies pour chaque requête.
– Possibilité de vérifier les fichiers télé versés.
– Injection de contenu dans la session HTTP.
– Blocage d'attaques à base de signatures.

Naxsi :

Naxsi est un module WAF pour NginX le serveur web et le fameux reverse-proxy, Naxsi est open source, hautement performant et à entretien réduit des règles. Son but est d'aider à la sécurisation des applications Web contre les attaques de type SQL Injection, Cross Site Scripting, Cross Site Request Forgery, les inclusions de fichiers locaux et distants. La différence principale entre Naxsi et d'autres WAF est qu'il ne bloque pas les attaques à base de leurs signatures, mais il utilise un modèle plus simple où, au lieu d'essayer de détecter les attaques « connues », il détecte les éléments inattendus dans les requêtes HTTP. Chaque type d'élément inhabituel va augmenter le score de la demande. Si la requête atteint un score considéré comme « trop élevé », la requête sera refusée et l'utilisateur sera redirigé vers une page spéciale. Il fonctionne un peu comme un système antispam [25].

WebKnight :

AQTRONIX WebKnight est un pare-feu d'applicatif pour Microsoft IIS, Il est un filtre ISAPI qui sécurise votre serveur Web en bloquant les mauvaises requêtes. WebKnight est bon pour obtenir de ce qui suit : débordement de tampon, attaque par traverse de répertoire, les attaques par encodage de caractère, injection SQL, Blocage mauvais robots, hotlinking, brute force, Et bien plus encore ...

Dans une configuration par défaut, toutes les requêtes bloquées sont enregistrées et vous pouvez personnaliser en fonction de vos besoins. WebKnight 3.0 a obtenu l'interface d'administration Web où vous pouvez personnaliser les règles et effectuer des tâches d'administration, y compris les statistiques [30].

4. Critères d'évaluation d'un pare-feu applicatif (WAFEC)

À mesure que les attaques d'applications Web d'aujourd'hui augmentent et que leur niveau relatif de sophistication augmente, il est vital d'élaborer des critères normalisé pour l'évaluation des WAF. Le projet de critères d'évaluation de l'application Web (Web Application Firewall Evaluation Criteria) sert deux objectifs [32]:

- Aidez les intervenants à comprendre ce qu'est un WAF et le rôle dans la protection des sites Web.
- Fournir un outil permettant aux utilisateurs de prendre une décision éclairée lors de la sélection d'un WAF.

WAFEC est un projet mixte entre The Web Application Security Consortium (WASC) et OWASP, veillant à ce que les meilleurs esprits de l'industrie, à la fois ceux qui travaillent jour et nuit pour développer des WAF et ceux qui les utilisent, s'engagent à faire en sorte que WAFEC soit complet, précis et objectif, et permet d'évaluer techniquement le meilleur WAF pour son environnement en fonction de 9 critères :

1. Type d'architecture à déployer (pont, reverse-proxy, intégré, …)
2. Support d'HTTP et d'HTML (Versions, encodages,…)
3. Techniques de détection (signatures, techniques de normalisation du trafic, …)
4. Techniques de protection (brute force, cookies, sessions, …)
5. Journalisation (intégration NSM, type de logs, gestion des données sensibles, …)
6. Rapports (types de rapports, distribution, format, …)
7. Administration (politiques, logs, …)
8. Performance (nb de connexions/s, latences, …)
9. Support XML (WS-i intégration, validation XML/RPC, …)

5. Discussion :

Il est bien connu que les solutions commerciales sont fermées, elles ne sont pas mises à la critique ou à la correction par la communauté et généralement on ne peut faire que des tests du genre boîte noire, à la différence d'une solution open source comme ModSecurity qui est maintenu et soutenu par une large communauté et de plus de dix ans d'expérience. L'open source est une sorte de garantie pour continuer à progresser sur l'échelle technologique, surtout que ce travail est censé être une étude, dont le but de mettre en place une solution efficace pour mitiger les risques et étendre ces fonctionnalités.

Les WAF nécessitent beaucoup de ressources pour analyser chaque en-tête et corps de requêtes et réponses, résultat nous devons penser à l'évolutivité si on veut pouvoir protéger tout notre trafic surtout s'il s'agit d'une monté en charge. Modsecurity est un produit logiciel open source, donc coté performances il est personnalisable, on peut améliorer le rendu de cette solution par rapport aux autres. Pour offrir une meilleure expérience avec ce pare-feu il suffit juste de combiner plusieurs instances avec un répartiteur de charges, ce qui va offrir à la foi plus de performance ainsi qu'une tolérance a la panne.

Une large communauté, une flexibilité, et personnalisable rend modsecurity très populaire ainsi très accessible à la documentation par sa grand communauté, surtout qu'il bénéficie d'un soutien des organismes comme OWASP ou un kit de règles de base (CRS) est fourni et régulièrement mis à jour par sa société en collaboration avec ces organismes.

6.Modsecurity

C'est un outil pour le suivi des applications web en temps réel, l'enregistrement et le contrôle d'accès. Il n'y a pas de règles qui vous dit ce qu'il faut faire; au contraire, il est à vous de choisir votre propre chemin à travers les fonctionnalités disponibles [33].

6.1 Fonctionnalités

La liberté de choisir ce qu'il faut faire est un élément essentiel de l'identité de ModSecurity et va très bien avec sa nature open source. Avec un accès complet au code source, votre liberté de choisir étend la possibilité de personnaliser et d'étendre l'outil lui-même pour l'adapter à vos besoins [33]. Ce qui suit est une liste des scénarios d'utilisation les plus importants:

- **le suivi de la sécurité des applications en temps réel et le contrôle d'accès**

À la base, ModSecurity vous donne accès au flux de trafic HTTP, en temps réel, ainsi que la capacité de l'inspecter. Cela suffit pour la surveillance de la sécurité en temps réel. Il y a une dimension supplémentaire de ce qui est possible à travers le mécanisme de stockage persistant de ModSecurity, ce qui vous permet de suivre les éléments du système au fil du temps et effectuer la corrélation d'événements. Vous êtes en mesure de bloquer de manière fiable, car ModSecurity utilise la requête et la réponse complète (mise en mémoire tampon).

- **la journalisation du trafic HTTP complet**

Les serveurs Web font traditionnellement très peu quand il s'agit de la journalisation à des fins de sécurité. Elles ne sont pas en mesure d'obtenir tout ce dont vous avez besoin. ModSecurity vous donne la possibilité de journaliser tout ce que vous avez besoin, y compris les données des transactions brutes, ce qui est essentiel pour le forensics. De plus, vous pouvez choisir quelles transactions seront enregistrées, quelles sont les parties d'une transaction sont enregistrées, et quelles parties sont assainis.

- **évaluation passive et continue de la sécurité**

Évaluation de la sécurité est largement considérée comme un événement programmé actif, dans lequel une équipe indépendante provient d'essayer d'effectuer une attaque simulée. L'évaluation passive et continue de la sécurité est une variation de surveillance en temps réel, où, au lieu de se concentrer sur le comportement des parties externes, vous vous concentrez sur le comportement du système lui-même. Il est un système d'alerte précoce de toutes sortes qui peuvent détecter des traces de nombreuses anomalies et les faiblesses de sécurité avant qu'elles ne soient exploitées.

- **durcissement de l'application Web**

L'un des caractéristiques principale de ModSecurity est la réduction de la surface d'attaque, dans laquelle vous affinez sélectivement les fonctionnalités HTTP que vous êtes prêt à accepter (par exemple, les méthodes de demande, les en-têtes de demande, les types de contenu, etc.). ModSecurity peut vous aider à renforcer et faire de nombreuses restrictions, soit directement, soit en collaboration avec d'autres modules Apache. Ils tombent tous sous le

durcissement de l'application Web. Il est possible de résoudre de nombreux problèmes de gestion de session, aussi bien que des vulnérabilités cross-site request forgery.

- **Quelque chose de petit, mais très important**

la flexibilité de ModSecurity est très pratique il peut aussi être quelque chose de complètement différent. Par exemple, certaines personnes utilisent ModSecurity comme un routeur de service Web XML, combinant sa capacité à analyser XML et appliquer des expressions XPath avec sa capacité à des demandes de procuration.

6.2 Les Phases de filtrage de Modsecurity

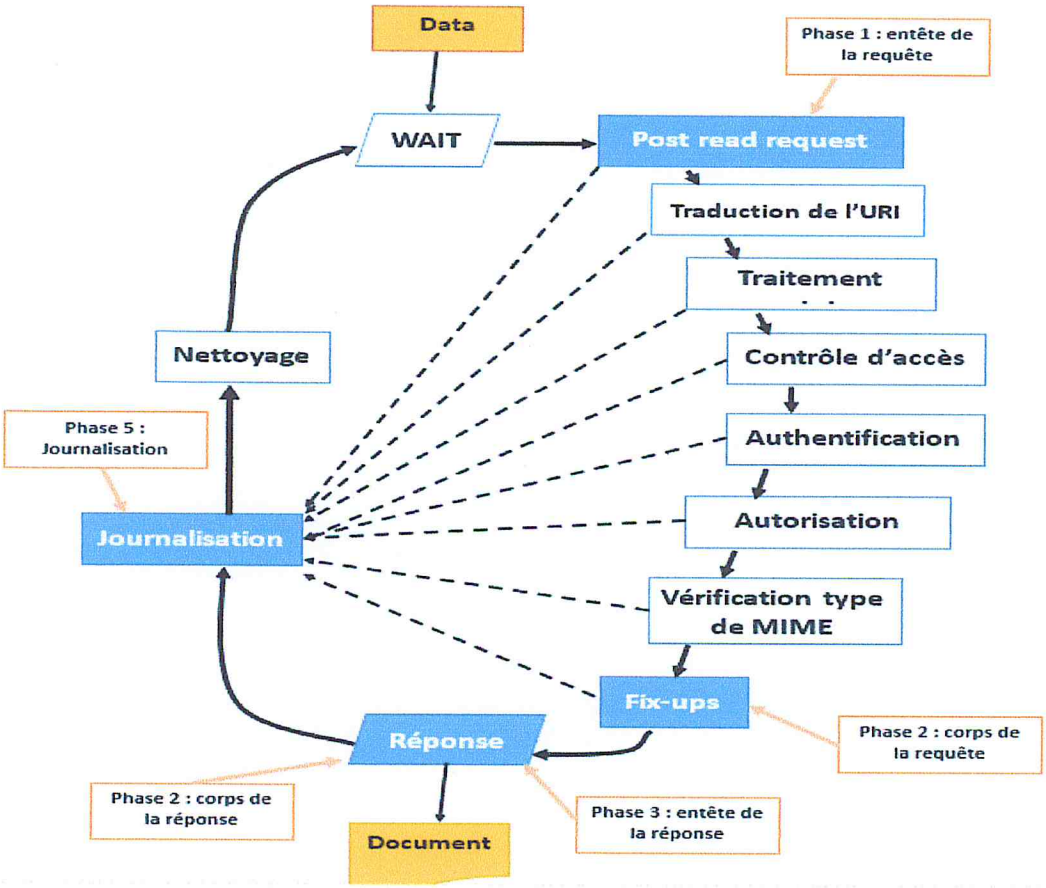


Figure 20 - les 5 phases de filtrage de Modsecurity [34].

Le filtrage peut intervenir non seulement sur les entêtes et corps de requête, mais aussi sur les entêtes et les corps des réponses. Toutes ces « interventions » peuvent être journalisées. L'administrateur peut créer des règles et les appliquer au niveau qu'il souhaite. La priorité d'application d'une règle est fonction d'une phase. Une règle de phase n est prioritaire par rapport à une règle de phase n+1.

Phase 1: entête de la requête

Dans cette phase Apache vient juste de lire l'entête de la requête, Il n'a pas encore lu son contenu, Il ne connaît pas encore les arguments contenus dans la requête, Toute règle s'exécutant en phase 1 intervient avant qu'Apache soit en mesure de faire quelque chose. Exemples de règles en phase 1:

- Décider si le corps de la requête doit être traité plus en détail.
- Définir comment le corps doit être traité(en XML ?)

Pour utiliser des règles ModSecurity en corrélation avec des espaces Web définis par Apache, celles-ci doivent être définies en phase 2.

Phase 2: corps de la requête

A cette étape Nous disposons désormais des arguments de la requête, Les règles de filtrage ou de rejets liées à des applications doivent intervenir à ce niveau.

Phase 3 : entête de la réponse

Cette phase intervient juste avant que les entêtes des réponses parviennent au client, Les règles de filtrage permettent de définir ce qu'il doit advenir de la « future » réponse, On décide si l'on veut analyser le contenu/corps de la réponse ou la bloquer, A ce niveau, nous ne savons pas encore ce que le serveur Apache va retourner.

Phase 4 : corps de la réponse

A ce niveau, ModSecurity analyse les informations renvoyées à destination du client. Le contenu HTML est analysé pour détecter :

- des fuites d'informations.
- des messages d'erreurs.
- des traces d'authentifications ayant échouées.
- etc.

Phase 5: journalisation

Les règles déclarées à ce niveau interviennent seulement sur la manière dont la journalisation doit s'opérer, Cette phase peut être utilisée pour analyser les messages d'erreur enregistrés par Apache, Elle permet également d'inspecter des entêtes de réponse qui n'étaient pas accessibles en phases 3 et 4, il est donc évident qu'il est trop tard pour interdire ou bloquer des connexions.

6.3 Fonctionnement des règles selon ModSecurity

ModSecurity est un pare-feu d'application Web assez puissant et polyvalent. Cependant, pour pouvoir l'utiliser, il est indispensable d'apprendre à dire à ModSecurity ce qu'on veut faire, à travers un langage propre à lui,

NB: *Un petit extrait du principe du fonctionnement des règles de modsecurity est attaché à l'annexe*

6.4 Les règles de base de ModSecurity – Core Rule Set (CRS)

ModSecurity est un moteur de pare-feu d'application Web qui offre très peu de protection à part entière. Pour cela, un kit de règles de base (ModSecurity Core Rules) nous est fourni. Pour être utile, ModSecurity doit être configuré avec des règles. Afin de permettre aux utilisateurs de profiter pleinement de ModSecurity, SpiderLabs de Trustwave a créé le projet OWASP ModSecurity Core Rules Set (CRS), Il est régulièrement mis à jour et testé par l'éditeur. Contrairement aux systèmes de détection et de prévention des intrusions, qui reposent sur des signatures spécifiques aux vulnérabilités connues, les CRS fournissent une protection générique contre les vulnérabilités inconnues souvent trouvées dans les applications Web, qui sont généralement codées sur mesure [33].

6.4.1 Contenu Règles de base (core rules set)

ModSecurity propose des règles par défaut qui luis sont fournie par OWASP, Elles sont basées sur une approche de type liste noire, et elles offrent une protection honorable contre les attaques les plus connues :

- Protection HTTP - détection des violations du protocole HTTP et une politique d'utilisation définie localement.
- Protection et détection des attaques des applications web commun.
- Détection de l'activité de certains scanners ou bots.
- interception de tentatives d'accès à des chevaux de Troie.
- Elles permettent de modifier les messages d'erreurs renvoyés par le serveur.

NB: *Un petit extrait du core de base des règles (CRS) de modsecurity est attaché à l'annexe*

7. Conclusion

De par la richesse de ses fonctions de sécurité il est vrai qu'un pare-feu applicatif peu se rendre très utile voire obligatoire dans certaines situations mais il ne faut pas pour autant en négliger l'aspect sécurité lors de l'élaboration d'une application, le fameux « Secure by design ».

La sécurité se tienne de différentes manières, on a pu énumérer quelques solutions primordiale comme : les Frameworks applicatifs, les équipements IPS, mais également leurs limites. On s'est focaliser sur le fonctionnement de la solution adoptée qui est le pare-feu applicatif, On a étudié quelques projet WAF qui sont très prometteurs et tendent à combler la sécurité des applications Web, puis on s'est approfondi à expliquer l'outil adopté.

Chapitre 3

Politique de sécurité des applications web

1. Introduction

Le présent chapitre est un guide pour orienter les responsables de la sécurité des systèmes d'information pendant les différentes phases du cycle de vie d'une application web, cela sert à les mieux sécuriser à travers des règles et de recommandations de sécurité que doivent être respectées.

2. Analyse des besoins

Les besoins et des objectifs des sécurités doivent être clairement définis lors de l'élaboration du projet, Ces objectifs doivent être spécifiés et exigés dans les clauses de sécurité se rapportant aux phases d'exécution du projet.

2.1 Exigences de sécurité

Les exigences de sécurité des applications web sont définie comme suite selon SANS [40] et par d'autre organisation de SSI, chacun des sujets suivant devrait être discuté et évalué par le développeur :

1. **Authentification**
2. **gestion de session**
3. **Contrôle d'accès**
4. **Gestion des entrées et sorties**
5. **Gestion d'erreur**
6. **Journalisation**
7. **Connexions aux systèmes externes**
8. **Chiffrement**

2.2 Evaluation du risque

Identifier et documenter les risques auxquels l'application pourrait être confrontée, que ce soit sur les biens (infrastructures informatiques, données, etc.) ou sur les fonctions importantes fournies par cette application. A cet effet, chacun des sujets énumérés dans la section des exigences ci-avant devrait être considéré [35].

2.3 Phase de conception

Une fois les besoins de sécurité et les menaces identifiés, il convient de concevoir la sécurité de l'application Web, c'est-à-dire de définir précisément les mécanismes de sécurité qui sont en mesure de répondre aux exigences fixées. Un document de conception doit décrire formellement ces mécanismes, en fournissant une bonne visibilité sur la manière dont les menaces seront gérées. La conception devrait clairement préciser les versions des logiciels et la plateforme utilisée ainsi que tous composants de tierce partie.

2.4 Formation et sensibilisation

Tous les membres de l'équipe-projet doivent être d'une part, sensibilisés aux enjeux et risques de sécurité et d'autre part, formés aux mécanismes de sécurité de base :

- La maîtrise d'ouvrage doit être en mesure d'identifier les enjeux de sécurité pour exprimer les besoins
- Le chef de projet doit connaitre les normes de sécurité à respecter lors du déploiement ;
- Les développeurs doivent être en mesure d'implémenter les règles de sécurité générales ainsi que celles spécifiques aux technologies Web utilisés ;
- En cas de développement externe, le prestataire doit justifier le fait qu'il dispose des compétences en développement sécurisé pour répondre convenablement au besoin. Malgré le respect des bonnes pratiques lors du développement des applications web, de nouveaux types de vulnérabilités peuvent apparaitre.

Il est donc important d'assurer une veille de sécurité pour remédier à ces vulnérabilités et éventuellement revoir les mécanismes de sécurité initialement mis en place.

3. Recommandations générale liés à l'infrastructure de l'application web

Avant d'entamer les précautions durant le développement une première série de précautions peuvent être prises au niveau de l'infrastructure (matériel et logiciel) hébergeant le service car un grand nombre d'attaques sont dues à la négligence de la sécurité lors du déploiement d'une application web.

La non suppression des dossiers d'installation, l'absence d'une politique de mise à jour, l'utilisation de composantes vulnérables, l'absence de segmentation et ségrégation du réseau, l'adoption d'une architecture réseau non sécurisé, etc. sont autant de problèmes qu'il faut absolument prendre au sérieux [38].

3.1 Sécurité physique et environnementale :

Une zone sécurisée : Les périmètres de sécurité doivent être définis et utilisés pour protéger les zones qui contiennent des informations sensibles ou critiques et des installations de traitement de l'information. Ces zones devraient être protégées par des contrôles d'entrée appropriés afin de garantir que seul le personnel autorisé soit autorisé à accéder.

La protection physique contre les catastrophes naturelles, les attaques malveillantes ou les accidents devraient être conçus et appliqués [36].

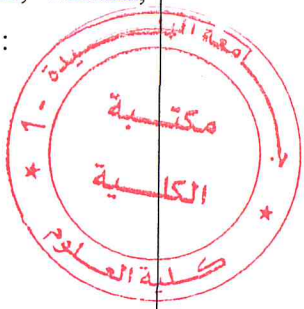
Les équipements : Pour éviter les pertes, les dommages, le vol ou le compromis des actifs, l'équipement doit être installé et protégé des risques liés aux menaces et aux dangers environnementaux et aux possibilités d'accès non autorisé. Aussi l'équipement doit être protégé des pannes d'alimentation et d'autres interruptions causées par des pannes dans le support des utilitaires. Tous ce qui est électricité, télécommunications, approvisionnement en eau, gaz, ventilation et climatisation Devrait être conforme aux spécifications du fabricant de l'équipement et aux exigences légales locales [36].

3.2 Administration :

Les mécanismes d'administration des sites web sont par nature des éléments sensibles dont l'exposition doit être limitée [39]:

R'1	Pour administrer les applications web via des protocoles sécurisés, Il est recommandé d'utiliser les protocoles sécurisés tels que SSH, SFTP et HTTPS.
R'2	Il est préférable de restreindre l'accès à l'administration des sites web aux seuls postes d'administration autorisés. Il est souhaitable de doter le serveur web d'une autre interface réseau réservée pour les connexions d'administration.

R'3	Eviter d'exposer les interfaces d'administration (par exemple : php-myadmin, webmin, cpanel.) à l'extérieur. Au cas où cette solution est nécessaire, il est important de : – Utiliser un VPN avec chiffrement – Utiliser des mots de passe fort – Activer un système de connexion à la demande – Journaliser l'activité des accès distants – Contrôler des tentatives d'accès.



Dans le cas d'un hébergement externalisé, il est recommandé d'administrer exclusivement l'application web qu'à partir d'une adresse IP fixe dédié.

3.3 Endurcissement de l'infrastructure

Il s'agit essentiellement de mettre en place certaines bonnes pratiques et apporter des modifications aux paramètres de la plateforme web afin de rendre difficile la tâche de compromission de l'application web. A cet effet, il faut [39]:

R'4	Installer le serveur Web sur une machine dédiée. L'installation d'autres services (DNS, SMTP, BD. . .) avec le service web sur le même serveur est à proscrire ;
R'5	Désinstaller les services et les composantes inutiles ;
R'6	Utiliser un disque dur physique dédié ou au moins une partition différente pour le contenu du site Web
R'7	Réserver une partition dédiée pour les fichiers « uploadés » par les utilisateurs ;
R'8	Personnaliser toutes les pages d'erreurs de telle façon à ne divulguer aucune information qui puisse aider un hacker ;
R'9	Supprimer les traces et configurer le serveur de manière à ne pas divulguer les technologies et les versions utilisées. Exemples : – Apache : mettre la variable server « tokens » dans le fichier « http.conf » à la valeur « prod » et la variable « server signature off ». – php : mettre la variable « expose_php » à la valeur « off » dans le fichier » php.ini ».
R'10	Réécrire des URLs pour cacher les technologies utilisées.
R'11	Supprimer du système d'exploitation les comptes issus de l'installation par défaut.

R'12	Supprimer toute la documentation fournie, les pages et applications d'exemples « sample applications », les dossiers d'installation, les dossiers de backup et les dossiers inutiles sur le serveur ;
R'13	Utiliser des outils de vérification de l'intégrité des fichiers au niveau du serveur web
R'14	Installer une solution antivirale et la mettre à jour ;
R'15	Tester les mises à jour sur une copie du serveur principale en vue de les valider avant de leur installation définitive sur le serveur de production ;
R'16	Respecter le principe du moindre privilège lors de la définition des droits des utilisateurs et l'exécution des services (exemple : enlever le droit d'exécution dans la partition "upload fichier") ;
R'17	Désactiver l'affichage du contenu des répertoires. Pour désactiver la fonction d'indexation des répertoires dans le serveur Apache, par exemple, il faut effacer « Indexes» dans la ligne « Option » du répertoire concerné au niveau du fichier « httpd.conf »
R'18	Interdire toute possibilité de remonter au répertoire racine afin d'empêcher la possibilité de redescendre vers les répertoires systèmes et exécuter des commandes non autorisées.
R'19	Renforcer la sécurité des différents OS, serveurs, technologies et logiciels en appliquant les guides d'endurcissement de sécurité relatifs à chaque produit.

3.4 Gestion des sauvegardes

Les procédures de sauvegarde et de récupération des données doivent être formalisées. Les configurations des différentes composantes de la plateforme web doivent être sauvegardées et stockées dans un coffre. Ce coffre doit être placé dans un local autre que celui qui abrite les serveurs afin d'éviter les dommages. Ces sauvegardes doivent être mises à jour et périodiquement vérifiées pour s'assurer du bon fonctionnement des médias de stockage [39].

3.5 Gestion de mise à jour

Pour assurer une gestion efficiente des patches « correctifs » de sécurité, il est important de :

R'20	Utiliser des outils automatisés pour la détection des patches de sécurité manquants (ex. utiliser Microsoft Baseline Security Analyzer pour les systèmes d'exploitation Windows) ou assurer une veille pour être au courant des nouveaux patchs
R'21	Vérifier la source du patch avant l'installation
R'22	Examiner attentivement les documents se rapportant au patch avant son application et Installer les correctifs dans un environnement de test avant de l'appliquer sur l'environnement de production.

En général, appliquer les patchs le plus tôt possible.

4. Recommandations au cours de développement

Cette partie liste les pratiques permettant de sensibiliser et aider les équipes de développement à créer des applications plus sécurisées. C'est une étape vers la construction d'une base de connaissances autour de la sécurité des applications web. Cette liste permet d'identifier les règles minimales nécessaires pour neutraliser les failles dans les applications les plus critiques. Le développeur doit suivre un ensemble de lignes directrices de codage sécurisé et d'utiliser un ensemble de bibliothèques de sécurité. L'équipe de développement peut se référer au guide de conception et d'implémentation d'applications Web sécurisées de l'OWASP[35].

4.1 Authentification et gestion de session

Les fonctions applicatives relatives à l'authentification et à la gestion de session ne sont souvent pas correctement mises en œuvre. Ceci permet aux attaquants de compromettre les mots de passe ainsi que les clés et les jetons de session, ou d'exploiter d'autres failles d'implémentation pour s'approprier les identités d'autres utilisateurs.

Authentification [40] **:**

R1	Ne pas coder en dur les identifiants c.-à-d. ne jamais laisser les informations d'identification stockées directement dans le code de l'application. CWE-798 [37]
R2	Développer un système de réinitialisation des mots de passe fort. Ces systèmes sont en général basés sur les réponses données par les utilisateurs suite à des questions personnelles et qui permettent de deviner leurs identités et réinitialiser leurs mots de passe. Le système doit être basé sur des questions qui sont à la fois difficiles à deviner et à casser. En outre, une option de réinitialisation de mots de passe ne doit pas révéler si un compte est valide ou pas afin d'empêcher l'énumération des noms des utilisateurs. CWE-640
R3	Gérer correctement les messages d'erreurs d'authentification afin de prévenir l'énumération des utilisateurs. A titre d'exemple, les messages d'erreurs qui révèlent que l'identifiant de l'utilisateur est valide et que le mot de passe est incorrect confirme que ce compte existe sur le système.
R4	Implanter le verrouillage des comptes contre les attaques de force brutale : utilisation des captcha. CWE-307
R5	Mettre en œuvre une politique complexe de mot de passe : – La longueur minimale d'un mot de passe doit être de 12 caractères au minimum ;

	– Les types de caractères qui doivent être utilisés dans un mot de passe sont : lettre (majuscule et minuscule), chiffre, caractère spécial ; – Le mot de passe ne doit pas comporter le login ; – Le nouveau mot de passe doit comporter au moins 4 différentes lettres par rapport à l'ancien mot de passe ; – Procéder au changement régulier des mots de passe ; – Une fois un mot de passe est expiré, cinq changements sont requis avant la réutilisation de ce mot de passe ; – La communication des mots de passe aux utilisateurs doit se faire d'une manière sécurisée. Il est recommandé de ne pas utiliser à cette fin un courrier électronique non protégé ; – Les mots de passe par défaut des constructeurs et des éditeurs doivent être modifiés après l'installation ; – Les comptes par défauts doivent être renommés ou désactivés. CWE-307 CWE-640
R6	Envisager pour des applications sensibles, l'utilisation d'authentification forte.

Gestion des sessions :

Le mécanisme de sessions permet de conserver des informations côté serveur entre deux requêtes du client. C'est ainsi que l'on peut, par exemple, gérer l'identification des utilisateurs. Une fois l'authentification réalisée, l'identité de l'utilisateur est associée à sa session pour pouvoir lui accorder les privilèges appropriés. Elle est définie par un jeton de session (ou identifiant de session) qui est généré aléatoirement et souvent enregistré du côté client dans un cookie. Dans ce cadre, il faut [40] :

R8	Les identifiants de session doivent être aléatoires et d'une entropie d'au moins 128 bits pour résister à l'analyse et à la prévision. CWE-6
R9	Régénérer les jetons de session : Les jetons de session doivent être régénérés chaque fois que l'utilisateur s'authentifie auprès d'une application ou change de niveau de privilège. De même, une fois le statut de cryptage change, le jeton de session doit obligatoirement être régénéré.
R10	Implémenter un délai d'attente de session inactive : Quand un utilisateur est inactif, l'application doit le déconnecter automatiquement. Les applications "Ajax" peuvent faire des appels récurrents à l'application qui réinitialise automatiquement le compteur d'inactivité. CWE-613

R11	Implémenter un temps d'expiration de la session : Après une période de connexion relativement longue (de 4 à 8 heures), les utilisateurs doivent être déconnectés. Ceci contribue à atténuer le risque qu'un attaquant utilise une session détournée pendant une longue durée. CWE-613
R12	Détruire la session si un signe d'altération est détecté : Si l'application nécessite plusieurs sessions simultanées pour un seul utilisateur, il est nécessaire d'implémenter des fonctionnalités pour détecter les tentatives de clonage de session. Une fois un signe de clonage est détecté, la session doit être détruite afin d'obliger l'utilisateur à s'authentifier de nouveau.
R13	Rendre la session invalide après la déconnexion : Lorsque l'utilisateur se déconnecte de l'application, la session et les données correspondantes sur le serveur doivent être détruites. Ceci garantit que la session ne soit accidentellement rétablie. CWE-613
R14	Placer un bouton de déconnexion sur chaque page : Le bouton ou le lien de déconnexion doit être facilement accessible sur chaque page après l'authentification.
R15	Utiliser des attributs de cookie sécurisés (les flags httponly et secure) : Le cookie de session doit être défini à la fois avec les flags HttpOnly et Secure. Ceci garantit que l'identifiant de session ne soit accessible aux scripts côté client et soit transmis via SSL respectivement. CWE-79 CWE-614
R16	Définir correctement le domaine et le chemin du cookie : Le domaine du cookie et l'étendue du chemin doivent être réglés sur les paramètres les plus restrictifs de votre application.
R17	Définir le délai d'expiration du cookie : Le cookie de session doit avoir une date d'expiration raisonnable si non, ces cookies doivent être évités

4.2 Protection des données

Dans beaucoup d'applications web, les données sensibles telles que les identifiants et les informations d'authentification ne sont pas correctement protégées. Ces données sensibles doivent être chiffrées en local et en transit. En général les précautions ci-après doivent être prises [40] :

R18	Stocker les mots de passe des utilisateurs avec une fonction de hash forte, itérative et salée : Les mots de passe utilisateurs doivent être stockés en utilisant des techniques de hachage sécurisées

	tel que l'algorithme SHA-256. En plus, plusieurs itérations de hachage et un sel (salt) aléatoire sont à implémenter afin de protéger efficacement un mot de passe. CWE-257
R19	Mettre en place des processus sécurisés de gestion des clés de cryptage et des certificats : Les clés doivent être correctement sécurisées lorsqu'elles sont stockées dans le système et ne doivent être accessibles qu'aux personnes habilitées. CWE-320
R20	Désactiver la mise en cache des données avec « des en-têtes de contrôle de cache » et le paramètre « autocomplete » : La mise en cache des données du navigateur doit être désactivée en utilisant les en-têtes de contrôle de cache HTTP ou les balises méta dans la page HTML. En outre, les champs de saisie sensibles (ex. formulaire de connexion) doivent avoir comme configuration dans le formulaire HTML « autocomplete = off » pour obliger le navigateur à ne pas mettre en cache les informations d'identification. CWE-524
R21	Effectuer une évaluation pour s'assurer que les données sensibles ne sont pas transportées ou stockées inutilement. Dans la mesure du possible, utiliser la tokenisation pour réduire les risques d'exposition aux données
R22	Affiner les droits d'accès de l'application web au serveur de base de données.

4.2.1 Protection des données en transit

La sécurisation des données sensibles durant leur transport est très importante. A cet effet, il faut [39] :

R23	Utiliser le protocole SSL : Le protocole SSL doit être utilisé à tous les niveaux de l'application. En cas de contraintes ou de limitation d'utilisation de ce protocole, il doit être appliqué à toutes les pages d'authentification ainsi que toutes les pages une fois l'utilisateur est authentifié. CWE- 311 319 523
R24	Désactiver l'accès HTTP pour toutes les ressources SSL activées : Pour toutes les pages nécessitant une protection par le protocole SSL, leurs URLs ne doivent pas être accessibles via un canal non-SSL. CWE-319
R25	Utiliser l'entête "Strict-Transport-Security" : L'entête "Strict-Transport Security" garantit que le navigateur ne communique pas avec le serveur via un canal non-SSL. Ceci permet de réduire le risque des attaques de type "SSL stripping" mises en œuvre par l'outil " sslsniff ".
R26	Echanger d'une manière sécurisée les clés de chiffrement : Si ces clés sont échangées ou prédéfinies au niveau de l'application, leur échange doit emprunter un canal sécurisé.
R27	Désactiver au niveau des serveurs les protocoles de chiffrements SSL faibles.

R28	Utiliser des certificats SSL validés par une autorité de certification digne de confiance et reconnue par le navigateur. Le nom sur le certificat doit correspondre au nom complet du domaine du site et la date d'expiration du certificat doit être encore valide.

4.3 Gestion des entrées et sorties

La validation des paramètres en « entrée » et en « sortie » requiert une importance capitale. L'équipe de développement doit faire très attention à ce niveau afin d'éviter des problèmes de sécurité non négligeables au niveau applicatif. A ce propos, les attaques les plus répandues et les plus dangereuses sont les attaques de type "XSS" et "injections SQL". Ces deux types d'attaques résident parmi les tops "10 d'OWASP".

En général, il est recommandé de créer un mécanisme centralisé de validation des entrées en tant que partie intégrante des applications web. Pour pallier aux problèmes précités, il est important de [39] :

R29	Préférer les listes blanches aux listes noires : Pour chaque champ de saisie, il faut appliquer un processus de validation de contenu. Le choix d'une liste blanche est privilégié. Le principe consiste à n'accepter que les données qui répondent à un certain nombre de critères. Parfois et pour plus de flexibilité, une liste noire est utilisée. Dans ce cas, il faut bloquer les mauvaises entrées de patterns et certains caractères utilisés dans les attaques d'injection. CWE-159 144
R30	Valider la source de l'entrée : La source de l'entrée doit être validée. Par exemple, si l'entrée est prévue à partir d'une requête POST, il ne faut pas accepter la variable d'entrée à partir d'une requête GET. CWE-20 346
R31	Utiliser des requêtes SQL paramétrées : Les requêtes SQL doivent être conçues de façon à véhiculer les entrées des utilisateurs à travers des variables de liaison afin de se prémunir contre les attaques de type "injection SQL". Il faut aussi interdire les requêtes SQL qui peuvent être créées d'une manière dynamique en utilisant la concaténation de chaînes de caractères. De même, la chaîne de requête SQL utilisée, liée ou paramétrée, ne doit jamais être construite de façon dynamique à partir des entrées d'un utilisateur CWE-89 56

R32	Utiliser des jetons pour empêcher les attaques de type "Cross-Site Request Forgery" (CSRF) : Il s'agit d'ajouter un paramètre obligatoire correspondant à un identifiant d'accès unique et non prédictible, régénéré pour chaque action utilisateur et par utilisateur (le plus efficace), ou pour chaque session (le moins efficace). Cet identifiant d'accès (ou "laisser-passer") est nommé "jeton CSRF". Ce mécanisme doit être implémenté sur chaque action qui dépend de l'utilisateur qui l'exécute (déconnexion, modification/suppression de données en BDD, etc.) et doit avoir une durée de validité limitée dans le temps. CWE-352
R33	Définir l'encodage de l'application : Pour chaque page de l'application, il faut définir l'encodage des en-têtes HTTP ou des méta-tags à l'intérieur de l'HTML. Le navigateur n'aura pas besoin de déterminer le type d'encodage. Un paramétrage cohérent d'un codage, comme UTF- 8, permet de réduire les attaques de type "Cross-Site Scripting" CWE-172
R34	Utiliser la Politique de Sécurité de Contenu (CSP) ou les en-têtes de protection "X-XSS" : ces deux techniques aident à se protéger contre les attaques réfléchies "cross-site scripting" (XSS).CWE-79 692
R35	Effectuer l'encodage contextuel des données produites (à la sortie) : Toutes les fonctions de sortie doivent contextuellement encoder les données produites avant de les présenter à l'utilisateur. La présentation des données produites « sortie » doit être codée différemment selon l'emplacement sortie dans la page HTML. Par exemple dans une page HTML, les données placées dans le contexte de l'URL doivent être encodées différemment de ceux placées dans le contexte "JavaScript". CWE-79
R36	Valider les fichiers soumis à l'application : Avant d'accepter une action "upload" des fichiers d'un utilisateur, il faut vérifier la taille, le type, le contenu et le chemin de destination du fichier. CWE-20 CWE-346
R37	Utiliser l'en-tête de réponse HTTP "X-Frame-Options" qui demande au navigateur de ne pas permettre l'utilisation de ' Frame ' d'autres domaines. Ceci pour empêcher le contenu d'être chargé par un site étranger dans un frame. Ceci atténue les attaques Clickjacking. CAPEC-103 CWE-693

4.4 Contrôle d'accès

A l'instar des vérifications des droits d'accès au niveau des applications web avant de rendre visible une fonctionnalité donnée sur l'interface de l'utilisateur, les mêmes contrôles doivent s'effectuer sur le serveur pour autoriser l'accès à une fonction donnée. Sinon, les attaquants peuvent forger des demandes pour accéder à une fonctionnalité non autorisée. A titre d'exemple, l'attaquant peut outrepasser (by passe) la phase d'authentification et accéder

directement à la page souhaitée en tapant directement son URL. Pour éviter ce genre de problèmes, les contrôles d'accès ci-après sont nécessaires [38] :

R38	Appliquer systématiquement les vérifications de contrôle d'accès : Pour garantir le déclenchement des contrôles d'accès d'un utilisateur qui s'est authentifié ou pas, il faut toujours appliquer le principe de la médiation complète. Cwe-284
R39	Appliquer le principe du moindre privilège : Toute décision d'accès à une donnée ou à une ressource doit répondre au principe du moindre privilège. Si elle n'est pas explicitement permise, l'accès doit être refusé. Si un utilisateur n'a d'autres besoins que de lire une colonne dans une table spécifique, l'administrateur de base de données ne doit lui octroyer que ce droit. Cwe-272 Cwe-250
R40	Accorder le minimum de privilèges pour le fonctionnement des applications et des middlewares : Si une application est compromise, il est important que cette application ainsi que tous les services de middleware ne puissent exécuter des taches autres que celles qui lui ont été attribuées. A titre d'exemple, si les couches "application ou métier" ont la capacité de lire et d'écrire des données dans une des bases de données, ces droits ne doivent pas lui permettre de toucher à d'autres tables ou bases de données.
R41	Ne pas utiliser les références d'objet directes pour les contrôles d'accès : Ne pas laisser les références directes à des fichiers ou des paramètres qui peuvent être manipulés pour accorder l'accès excessif. Les décisions de contrôle d'accès devraient être basées sur l'identité de l'utilisateur qui s'authentifie ainsi que sur l'information de confiance du côté serveur. Cwe-284
R42	Ne pas utiliser des renvois ou des redirections non validés : Une redirection non validée peut permettre à un attaquant d'accéder sans authentification au contenu privé. En outre, les redirections non validées permettent à l'attaquant de renvoyer les victimes vers des sites malveillants. Il faut empêcher l'apparition de ces redirections en effectuant les vérifications de contrôles d'accès appropriés. Pour les redirections dynamiques, adopter un fonctionnement en liste blanche en vérifiant que les URL visées soient légitimes. Cwe-601

4.5 Gestion des erreurs

La gestion des erreurs garantit qu'en cas d'erreur, aucune information importante n'est présentée à l'utilisateur. Il est à noter, que lors de la phase de reconnaissance, un attaquant

stresse l'application pour recueillir le maximum d'information à travers les messages d'erreurs. Pour s'y protéger, ci-dessous les recommandations à suivre [39] :

R43	Afficher des messages d'erreurs génériques : Les messages d'erreurs ne doivent pas révéler des détails sur l'état interne de l'application. Par exemple, il ne faut jamais exposés à l'utilisateur via des messages d'erreurs le chemin du système de fichiers ou la pile d'informations. Cwe-209
R44	Gérer toutes les exceptions : Les gestionnaires d'erreurs doivent être configurés pour gérer les erreurs inattendues et contrôler minutieusement toutes les sorties possibles. Cwe-391
R45	Gérer les erreurs générées par les Framework : la plateforme de développement peut générer des messages d'erreurs par défaut qui révèlent parfois des informations importantes. Ces messages devraient être remplacés par des messages d'erreurs personnalisés. Cwe-209

4.6 Journalisation

La journalisation est un composant important de la sécurité d'une application web. A cet effet, il est nécessaire de définir une politique de journalisation précisant notamment les modalités et les durées de conservation des différents journaux. Une bonne gestion des journaux passe par [39] :

R46	L'enregistrement de toutes les activités d'authentification dans les logs : Toute activité d'authentification, réussie ou non, doit être enregistrée. Cwe-778
R47	L'enregistrement de toutes les modifications des droits dans les logs : Toute activité de changements de niveau de privilège d'un l'utilisateur doit être enregistrée. Cwe-778
R48	L'enregistrement de toutes les actions d'administration dans les logs. Cwe-778
R49	L'enregistrement de tout accès à des données sensibles dans les logs Cwe-778
R50	L'interdiction de l'enregistrement des données confidentielles (mot de passe, numéro de carte bancaire..) dans les logs. Pour des raisons d'audit, ces données doivent être enregistrées d'une manière chiffrée. Cwe-778
R51	L'enregistrement du contenu des champs "referer et user-agent" pour assurer une traçabilité de toutes les activités. Cwe-532
R52	Stocker les logs en toute sécurité : Cwe-533

R53	Les journaux doivent être lisibles et facile à analyser via des scripts : utiliser des délimiteurs, des types et des formats de données bien définis Cwe-778
R54	Le groupe date-heure du journal est important pour l'analyse des logs. Le groupe date-heure doit être inscrit au début de la ligne, en deux formats : un format « epoch » et un format lisible par les utilisateurs tout en indiquant le fuseau horaire, de préférence GMT ou UTC. Cwe-778
R55	Chaque log doit avoir un identifiant. Cwe-778
R56	Pour faciliter le filtrage, une ligne de log doit contenir le maximum d'informations utiles et éviter les logs multi-lignes. Cwe-778

5. Conclusions :

Ce chapitre étais accorder à une politique de sécurité où nous avons cité un ensemble d'exigences de sécurité, nous avons proposé des recommandations et de bonne pratique pour s'y protéger. Nous concluons ce chapitre en disons que bien sécuriser un site web dans sa phase de développement est essentielle, les programmeurs et les administrateurs doivent comprendre parfaitement les risques pour s'y faire face et pour que l'application tienne toutes ses promesses de sécurité.

Chapitre 4

DÉMARCHE ET CONCEPTION

1. Introduction

Ce chapitre comportera l'architecture qui compose le système adopté, objectivement nous tenons à ajouter une couche de protection via un pare-feu web, cela à travers des règles de filtrage du trafic avant que celui-ci n'atteigne l'application web, puis nous présentons une modélisation d'un module d'administration et de suivi des évènements afin d'étendre les fonctionnalités proposées.

2. plateforme

Les pare feu applicatifs web permettent de mettre en œuvre des politiques de sécurité adaptées à chaque application, une démarche qui devra être toujours complétée par un code source sécurisé. Placé devant une application, en temps réel le WAF surveille alors le trafic avant qu'il atteigne l'application web, et analyse toutes les requêtes grâce à une base de règles afin d'éliminer par filtrage tout schéma de trafic potentiellement préjudiciable. Nous comptons utiliser cette solution pour placer une couche de contrôles sécuritaire avant que le trafic aborde le code source de l'application web.

Cette partie comportera l'architecture logicielle et matérielle de la plateforme pour déployer un système de protection. Nous avons rajouté aux fonctionnalités du pare-feu web un module qui comporte une interface de monitoring offrant un retour visuel en temps réel sur les évènements, et un module d'administration du pare-feu applicatif pour compléter l'ergonomie.

L'architecture logicielle et matérielle de cette plateforme est montrée dans le schéma suivant :

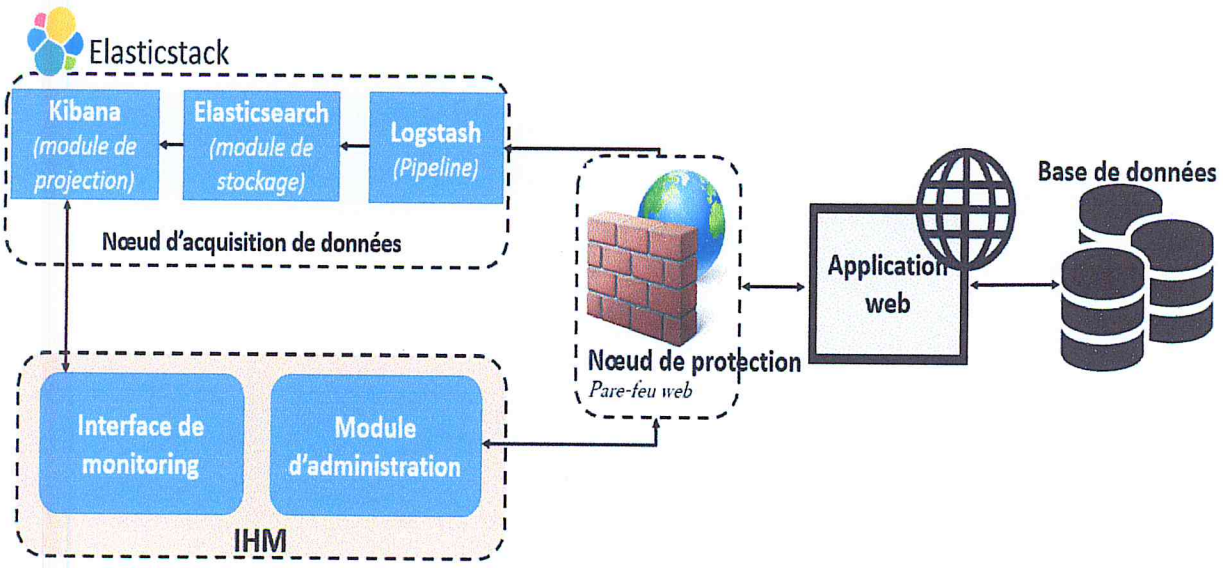


Figure 21 – l'architecture de la plateforme proposée (Le module rajouté est coloré en rose)

L'architecture de l'infrastructure se compose de trois parties :

- Le pare-feu web qu'on va appeler « nœud de protection », c'est le nœud à travers lequel nous appliquons une certaine politique par des règles de filtrage.
- La suite elasticstack qu'on va appeler « nœud d'acquisition de données », c'est le nœud par lequel on va extraire des informations de puis le pare-feu applicatif, pour construire une base de données pertinente.
- L'interface homme machine, c'est le nœud qui nous permet de contrôler le pare-feu web

2.1 Nœuds de protection (WAF)

Nous entendons dire par nœud de protection notre pare-feu applicatif, qui sera en première ligne de défense devant l'infrastructure de l'application web, il reçoit les requêtes et procède à la détection de la présence d'un facteur suspect, sinon il redirige la requête vers le serveur de destination.

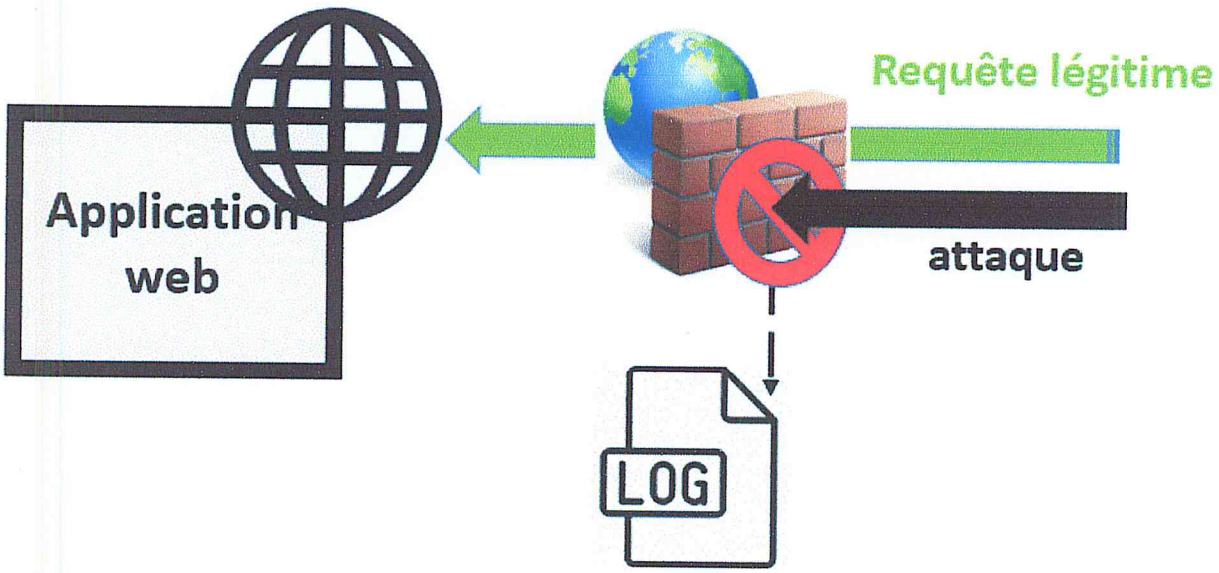
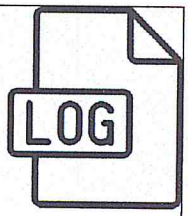


Figure 22 - illustration de fonctionnement du nœud de protection

Nous avons paramétré le pare-feu pour journaliser les requêtes bloqué dans un fichier. A l'intérieur de ce fichier on a déterminé ce qu'il faut enregistrer : entrée de journal (contient les informations de base : l'heure, l'ID unique … etc.), en-têtes de requêtes, en-têtes de réponses, corps de réponses, et les informations sur la transaction du pare-feu (si la demande a été autorisée ou refusée, le code d'état HTTP pertinent ainsi que le message du pare-feu).



--5759e83f-A--

[27/Mar/2017:14:22:32 +0000] dqIu7V5MziQAAEpPAWwAAAAE 94.76.206.36 38037 94.76.206.36 80

--7a19264b-B--

GET /a/a2/passage.php?login=admin%27+or+%271%27%3D%271%27%23&password=azzertt HTTP/1.1

Host: 127.0.0.1

User-Agent: nikto.

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://127.0.0.1/a/a2/

Connection: keep-alive

Upgrade-Insecure-Requests: 1

--5759e83f-F--

HTTP/1.1 403 Forbidden

Content-Length: 275

Connection: close

Content-Type: text/html; charset=iso-8859-1

--7a19264b-E--

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head>

<title>403 Forbidden</title>

</head><body>

<h1>Forbidden</h1>

You don't have permission to access /a/a2/passage.php on this server.

<hr>

<address>Apache/2.4.7 (Ubuntu) Server at 127.0.0.1 Port 80</address>

</body></html>

--7a19264b-H—

Message: Access denied with code 403 (phase 2). Pattern match "(?i:([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)\\b([\\d\\w]++)([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)(?:(?:=|<=>|r?like|sounds\\s+like|regexp)([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)\\2\\b|(?:!=|<=|>=|<>|<|>|\\^|is\\s+not ..." at ARGS:login. [file "/usr/share/modsecurity-crs/activated_rules/modsecurity_crs_41_sql_injection_ attacks.conf"] [line "77"] [id "950901"] [rev "2"] [msg "SQL Injection Attack: SQL Tautology Detected."] [data "Matched Data: '1'='1 found within ARGS:login: admin' or '1'='1'#"] [severity "CRITICAL"] [ver "OWASP_CRS/ 2.2.8"] [maturity "9"] [accuracy "8"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/ WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"]

Action: Intercepted (phase 2)

Apache-Handler: application/x-httpd-php

Stopwatch2: 1504389726726236 15448; combined=1100, p1=657, p2=427, p3=0, p4=0, p5=15, sr=25, sw=1, l=0, gc=0

Response-Body-Transformed: Dechunked

Producer: ModSecurity for Apache/2.7.7 (http://www.modsecurity.org/); OWASP_CRS/2.2.8.

Server: Apache/2.4.7 (Ubuntu)

Figure 23 - aperçue de la journalisation d'une seule attaque

2.1.1 Atténuation d'attaques via le WAF

Puisque la mission de ce nœud est une mission clef dans le système on va présenter une section pour une stratégie de mitigation d'attaques avec des règles de filtrage.

Évidemment certaines failles doivent être prises en compte au niveau de l'application, bien sûr on n'ouvre pas la porte aux bandits puis on se demande pourquoi on se fait voler. Un WAF est censé remédier à ces vulnérabilités étant le pont par lequel tout passe.

Présenter toutes les vulnérabilités n'est vraiment pas aisé dans notre contexte d'étude, car chacune des attaques nécessite en elle-même une recherche approfondie de ces usages théoriques et pratiques. C'est pourquoi nous expliquerons en bref les grandes lignes en tenant exemple de certaines de ces attaques et des approches de défense munies par le pare-feu. Les règles de politiques de nos configurations standards tirent profit de plusieurs signatures génériques comme OWASP ModSecurity Core Rule Set (CRS).

Bloqué les requêtes proxy

Les requêtes routées via les proxys peuvent être problème pour certains sites. Les utilisateurs peuvent se cacher derrière l'anonymat perçu d'un proxy serveur et lancer n'importe quoi, vous pouvez donc vouloir bloquer les requêtes soumises si vous trouvent qu'ils causent des problèmes sur votre site.

Une façon de le faire, c'est de vérifier la présence de l'en-tête X-Forwarded-For dans le requête HTTP. Si cet en-tête existe, cela signifie que la requête a été effectuée par un proxy serveur pour le compte de l'utilisateur réel.

Cette règle détecte et bloque les requêtes des serveurs proxy qui utilisent l'En-tête X-Forwarded-For:

```
SecRule &REQUEST_HEADERS: X-Forwarded-For "@gt 0" deny
```

La règle utilise l'opérateur '&' pour énuméré le nombre d'en-têtes de requête qui présentes le nom X-Forwarded-For. Si ce nombre est supérieur à zéro, cela signifie qu'un en-tête est présent et la requête est bloquée.

Les injections

Les attaques par injection se produisent lorsqu'une donnée non contrôlée est envoyée à un interpréteur dans le cadre d'une commande ou d'une requête afin d'exécuter des actions imprévues ou d'accéder à des données non autorisées.

les différentes attaques par injection reposent principalement sur le même principe qui est l'utilisation de mots clefs et de caractères spécifiques qui permettent par exemple de mettre en commentaire des portions de code et d'insérer du code frauduleux. Il est cependant rare que l'application ait besoin d'accepter tous ses éléments, comme :

```
` ' " \ * / = + ! & < > ~ # { [ ( ) ] } | ^ $ @ . , : ;
```

Anticiper ces attaques consiste donc à bien empêcher les requêtes porteuses de caractère spécial ou mot clé.

Cette règle détecte et bloque les requêtes qui contiennent des caractères qui peut s'agir d'une injection:

```
SecRule ARGS_GET "@contains (caractere dangeureu)" "deny,msg:'SQL Injection'"
```

Prévenir des attaques XSS

La mesure la plus importante que vous pouvez prendre pour éviter les attaques XSS est de s'assurer que toutes les données fournies par l'utilisateur qui sont affichées dans vos pages Web sont correctement désinfectées. Cela signifie que le remplacement des caractères potentiellement dangereux, tels que les parenthèses incluses (<et>) avec leurs versions codées de l'entité HTML correspondantes, dans ce cas < Et &gt.

Voici une liste de caractères que vous devez encoder lorsqu'ils sont présents dans des données fournies par l'utilisateur pour être incluses dans les pages Web :

caractères	HTML-encode
<	&lt ;
>	&gt ;
(	&#40 ;
)	&#41 ;
#	&#35 ;
&	&amp ;
''	&quot ;
'	&#39 ;

```
SecRule REQUEST_BODY "@detectXSS" "id:12345,log,deny"
```

Prévenir les attaques de force brute

Les attaques de force brutale permettent à un attaquant d'accéder à une ressource en essayant à plusieurs reprises de deviner des noms d'utilisateur, des mots de passe, des adresses de courrier électronique ou d'autres ressources critiques, elles peuvent être très efficaces si aucune protection n'est mise en place.

Une bonne façon de se défendre contre les attaques de force brute est de permettre un certain nombre de tentatives de connexion, disent trois, et après cela, commencer à retarder ou à bloquer d'autres tentatives. Voyons comment nous pouvons utiliser ModSecurity pour y parvenir.

Si votre page de connexion login est située à votresite.com/login, alors les règles suivantes suivront le nombre de tentatives de connexion des utilisateurs :

```
Bloquer les tentatives de connexion après 3 tentatives échouées
<LocationMatch ^/login>
# initialiser la collection IP avec l'adresse IP de l'utilisateur
SecAction "initcol:ip=%{REMOTE_ADDR},pass,nolog"

# Détecter les tentatives de connexion échouées
SecRule RESPONSE_BODY "Username does not exist" "phase:4,pass,setvar:
ip.failed_logins=+1,expirevar:ip.failed_logins=60"

# Bloquer les tentatives de connexion ultérieures
SecRule IP:FAILED_LOGINS "@gt 3" deny
</Location>
```

Les règles initialise la collection IP et augmenter les ip.failed_logins après chaque tentative de connexion échouée. Une fois plus de trois connexions échouées sont détectés, les prochaines tentatives sont bloquées. L'action **expirevar** est utilisée pour réinitialiser le nombre de tentatives de connexion échouées à zéro après 60 secondes, de sorte que le blocage sera en vigueur pendant une durée maximale de 60 secondes.

2.2 Nœud d'acquisition de données

Gérer les évènements générés par le pare-feu est la mission principale de ce nœud, dans cette section le but est de chercher, analyser, et structurer les données des fichiers log du pare-feu, pour qu'elles deviennent exploitables par des outils de programmation. A travers cette partie du système nous cherchons à récolter des données pertinentes et construire une base de données centralisé qui va être par la suite utilisé pour surveiller les évènements qui se produise dans une interface de monitoring, et dans le même contexte nous cherchons à développer une base de connaissance qui nous aide à améliorer la politique de sécurité.

On à exploiter la suite ElasticStack (Logstash, Elasticsearch, et Kibana) version 5.0 pour les échanges de données entre le pare-feu web et l'IHM, c'est une suite logicielle pour le traitement de données en temps réel. Nous avons choisi cette solution pour ça performance en matière de recherche et d'analyse de données structurées et non structurées, elle est beaucoup utilisé lors du passage à l'échelle (big data) pour son moteur de recherche puissant. Le rôle de ce nœud se résume dans ces étapes :

2.2.1 Logstash (pipeline)

Le rôle de ce module est de récupérer les données depuis une source où ils sont éparpillés puis procéder à l'analyse de ces données afin de les filtrer et structurer puis les transporter vers un module de stockage. D'où nous avons tiré profit de l'outil (logstash) pour pouvoir analyser et filtrer les logs de notre pare-feu web afin d'obtenir les données voulues. Nous utilisons logstash comme un pipeline entre la source de la donnée qui est le fichier de journalisation et le module de stockage (elasticsearch), il nous permet d'analyser, chercher, et extraire des données pertinentes de puis les fichiers logs. Le schéma de fonctionnement de logstash :

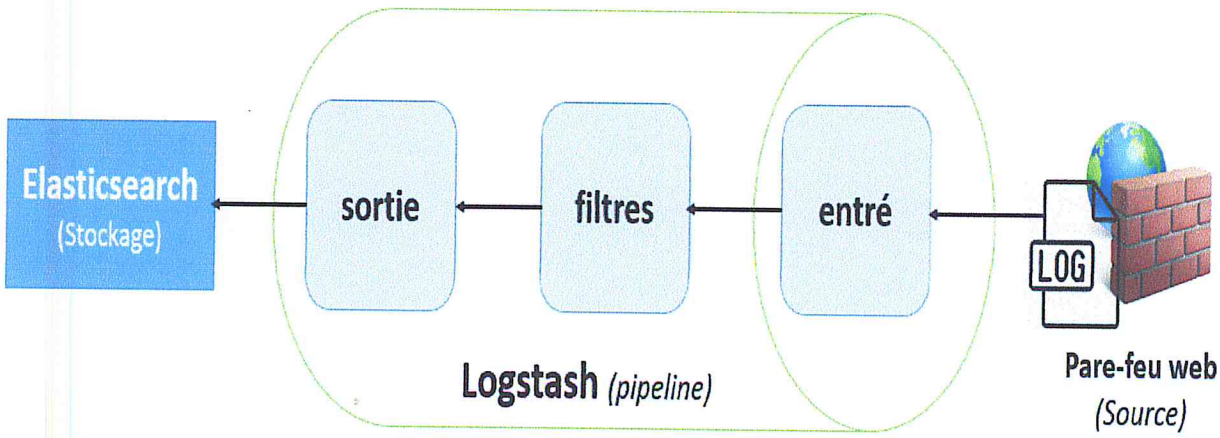


Figure 24 - fonctionnement du pipeline (logstash) dans notre architecture

Les traitements dans le pipeline (Logstash) sont composés en trois parties (entrée, filtre, sortie):

Entrée :

Au cours de cette partie on doit spécifier l'emplacement de l'ingestion des données. Dans notre cas c'est les fichiers logs de pare-feu web. Dans un second traitement on doit gérer correctement les logs qui sont un texte multi-lignes, on doit savoir comment dire quelles lignes font partie d'un seul événement. Il est évident que chaque début d'évènement commence avec cette étiquette `- - 5759e83f - A - -` qui est le marqueur d'entrée de chaque nouveau log d'évènement, où la partie souligné en rouge est variable, et inversement le reste est fixe. Pour déterminer les lignes de logs qui sont du même évènement nous avons mis en place une expression régulière qui identifie cette suite de caractères.

L'automate suivant montre le fonctionnement de l'analyse syntaxique qui identifier la chaine de caractères qui affirme un nouvel évènement :

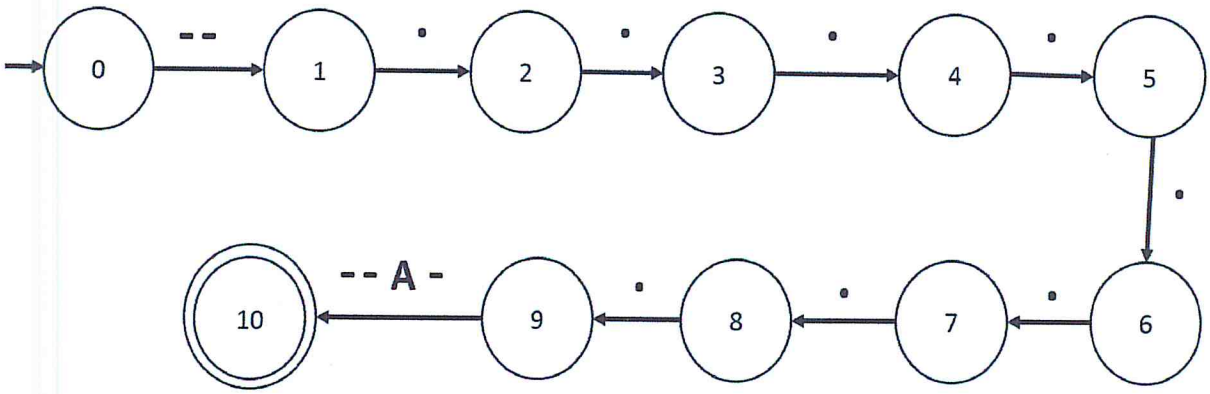


Figure 25 – automate qui détermine un nouvel évènement

Cela consiste à analyser les logs en constituants une chaîne composés des unités lexicales suivantes :

A : Constante alphabétique

- : Constante caractère spéciale

. : Variable qui signifie tout caractère alphanumérique

L'automate ci-dessus doit vérifier l'existence de la suite de caractère qui correspond à l'expression régulière de logstash : `^ - - . . . . . - A - -` , le symbole ^ signifie le début du texte ou de la ligne. Si la vérification est confirmé ça veut dire que les lignes suivante appartiennent au même évènement jusqu'à la réapparition d'une nouvelle expression similaire.

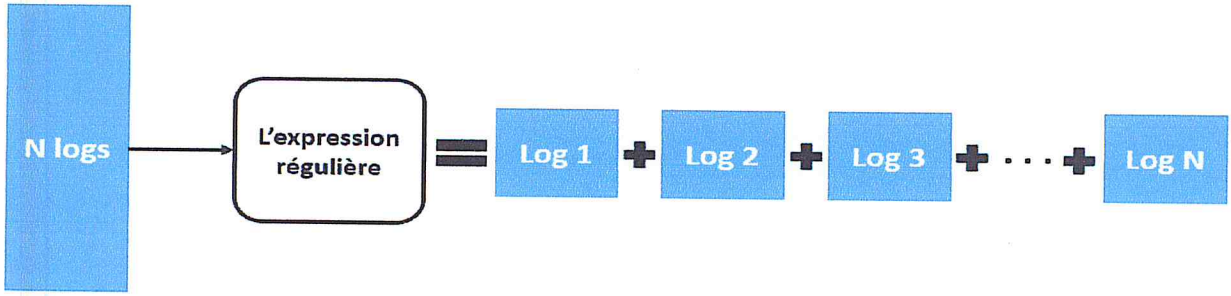


Figure 26 –Rôle de l'expression régulière

2. Filtre :

Le but de cette partie est d'analyser les logs pour filtrer les données bruités, et extraire les données pertinentes. Pour extraire ces données pertinentes nous avons adapté un modèle de correspondance (pattern matching) conforme au format du log d'un évènement d'attaque, et ainsi filtrer et garder que les données pertinentes que nous voulons exploiter. Cette étape permet la structuration des données éparpillées dans les fichiers logs.

La figure 28 est un organigramme qui résume l'étape de filtration du fichier de journalisation :

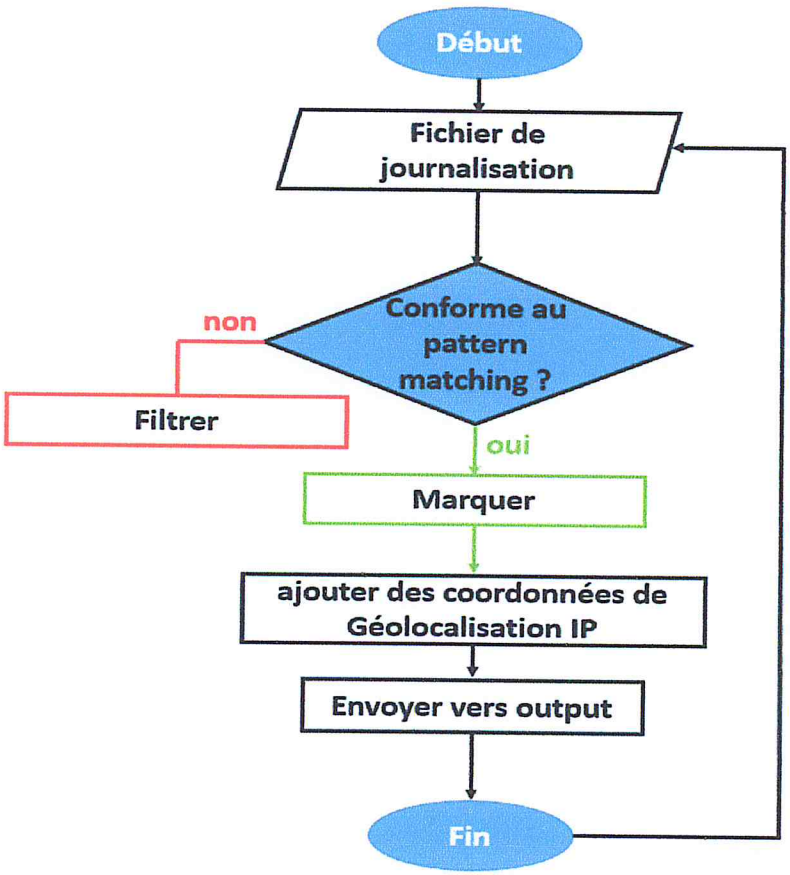


Figure 27 – organigramme du filtre des logs.

Extraction des donnés via Pattern matching :

Dans le filtre nous avons mis en place un pattern matching qui permet de trouver les données des logs souhaité. Dans ce sens nous avons utilisé **Grok** qui est une fonction de logstash qui permet de correspondre les logs avec des patterns et de les transformer en données structuré consultable.

Grok combine les patterns de texte prédéfinis avec le format qui correspond aux journaux.

La syntaxe d'un pattern **Grok** est %{SYNTAX : SEMANTIC}

SYNTAX : est le nom du pattern prédéfinis qui correspond à notre texte. Par exemple, 3.44 sera apparié par le modèle NUMBER et 55.3.244.1 sera le modèle IP. La syntaxe est la façon dont vous faites correspondre.

SEMANTIC : est l'identifiant (tague / marque) que nous donnons au morceau de texte correspondant. Par exemple, 3.44 pourrait être la durée d'un événement, donc vous pouvez l'appeler simplement la durée. En outre, une chaîne 55.3.244.1 pourrait identifier le client en faisant une demande.

Avec ce principe de syntaxe et sémantique, nous pouvons extraire des champs utiles à partir d'un journal, nous expliquons ce mécanisme avec un échantillon dans l'exemple suivant.

Exemple : 55.3.244.1 GET /index.html 15824

Pour cela le pattern pourrait être :

%{IP:client} %{WORD: méthode } %{URIPATHPARAM: requête } %{NUMBER:bytes}

Un exemple dans logstash:

```
filter {
grok {match =>
{"inputs" => "%{IP : client} %{WORD : méthode} %{URIPATHPARAM : requête} %{NUMBER:bytes} }
 }
}
```

Après le filtre grok, l'événement aura comme résultat les champs ci-dessous :

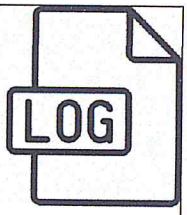
-client : 55.3.244.1

-méthode : GET

-requête : /index.html

-bytes : 15824.

Les parties colorées en jaune c'est les champs que nous souhaitons extraire pour les exploiter, le reste du log sera filtrer.



--5759e83f-A--

[27/Mar/2017:14:22:32 +0000] dqIu7V5MziQAAEpPAWwAAAAE 94.76.206.36 38037 94.76.206.36 80

--7a19264b-B--

GET /a/a2/passage.php?login=admin%27+or+%271%27%3D%271%27%23&password=azzertt HTTP/1.1

Host: 127.0.0.1

User-Agent: nikto.

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://127.0.0.1/a/a2/

Connection: keep-alive

--5759e83f-F--

HTTP/1.1 403 Forbidden

Content-Length: 275

--7a19264b-E--

<html><head>

<title>403 Forbidden</title>

</head><body>

You don't have permission to access /a/a2/passage.php on this server. </body></html>

--7a19264b-H—

Message: Access denied with code 403 (phase 2). Pattern match "(?i:([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)\\b([\\d\\w]++)([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)(?:(?:=|<=>|r?like|sounds\\s+like|regexp)([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)\\2\\b|(?:!=|<=|>=|<>|<|>|\\^|is\\s+not ..." at ARGS:login. [file "/usr/share/modsecurity-crs/activated_rules/modsecurity_crs_41_sql_injection_attacks.conf"] [line "77"] [id "950901"] [rev "2"] [msg "SQL Injection Attack: SQL Tautology Detected."] [data "Matched Data: '1'='1 found within ARGS:login: admin' or '1'='1'#"] [severity "CRITICAL"] [ver "OWASP_CRS/ 2.2.8"] [maturity "9"] [tag "WASCTC/ WASC-19"] [tag "OWASP_TOP_10/A1"] ....

Figure 28 - les parties de log souhaité

Nous avons choisie de extraire uniquement certain parties de logs pour deux raisons, sois car les parties extraites peut déterminer les données filtré, ou bien parce que c'est des informations sans bénéfices dans notre contexte d'étude (exp : version serveur, message d'erreurs … etc).

Sortie :

Après avoir filtré les données, logstash envoie les données extraites vers le module de stockage (elasticsearch) qui est chargé de les stockée d'une manière fiable et bien structurer.

2.2.2 Elasticsearch (module de stockage)

Nous comptons utiliser cet Elasticsearch comme un nœud de stockage qui permet d'indexer et stocker ces données de manière centralisée pour pouvoir les interroger en toutes circonstances. Cet outil est un moteur de recherche et d'analyse de données open-source hautement évolutif, il nous permet de rechercher de gros volumes de données rapidement et en temps quasi réel, il est utilisé comme moteur / technologie sous-jacent qui gère les exigences de recherche complexes.

Le but de ce stockage est de construire une base de connaissance exploitable qui va être affiché sur une interface de monitoring ainsi avoir une bonne image qui nous aide optimiser notre politique de sécurité. Le rôle de cet outil est illustré dans ce petit schéma :

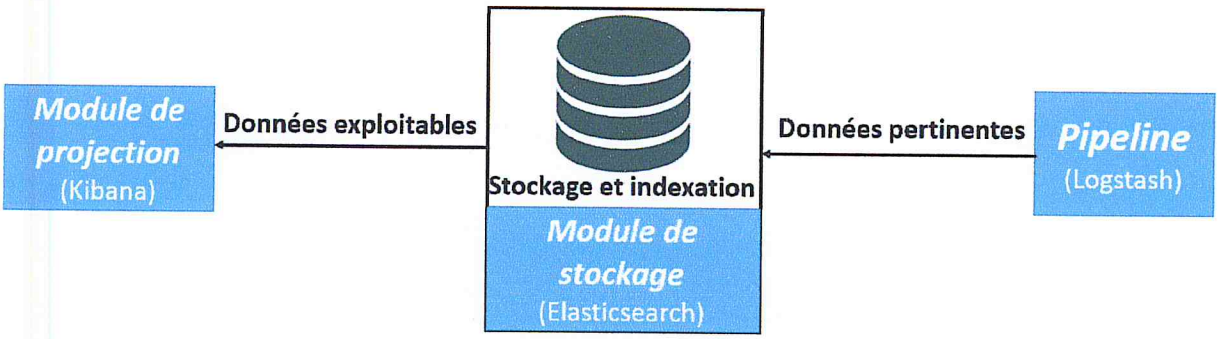


Figure 29 – Rôle du module de stockage (elasticsearch) dans notre plateforme

La table suivante montre l'ensemble des données pertinentes extraite de puis le fichier de journalisation du pare-feu web, ces données seront structuré, stocké et puis interroger.

***Tableau 6** – table de données pertinente extraites*

Données pertinentes	
- Id	- geoip.country_name
- timestamp	- geoip.continent_code
- User Agent	- geoip.city_name
- IP source	- geoip.ip
- port source	- geoip.location
- IP destination	- geoip.timezone
- destination Port	
- HOST	
- REFERER	
- Version http	
- méthode Http	
- requête	
- type d'attaque	
- gravité	
- rule id	
- rule data	
- rule file	

2.2.3 Kibana (module de projection)

Kibana est un outil qui permet de consulter Elasticsearch et d'explorer nos données stocké, on peut facilement projeter une analyse de nos données dans une variété de tableaux, courbes, figures ou cartes. Ces visualisations qui interrogeant les données qu'on a préalablement structuré et stocké, pour a la fin pouvoir les intégré dans notre interface de monitoring.

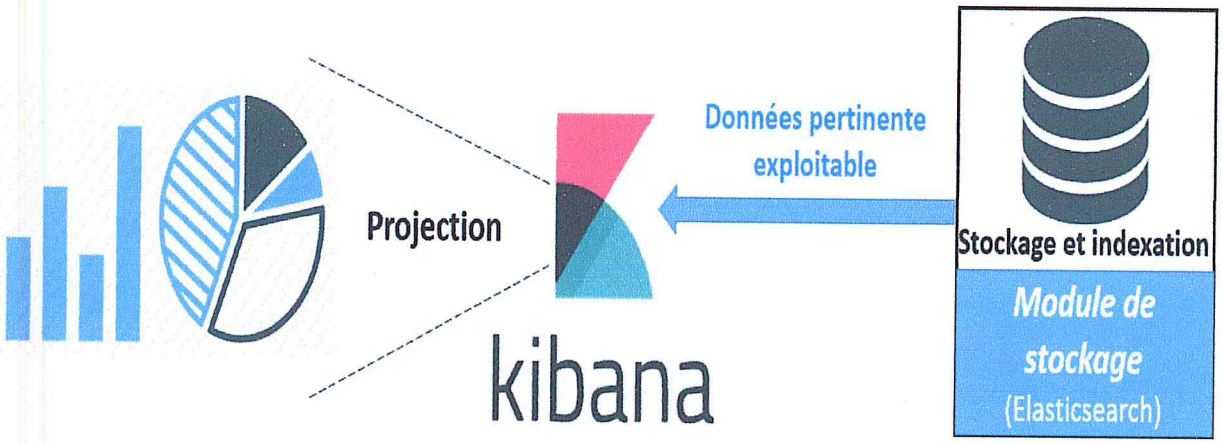


Figure 30 – Rôle de module de projection (Kibana) dans notre plateforme

2.3 L'interface homme machine

L'interface graphique d'administration et de monitoring doit supporter toute option de configuration fournie par le nœud de protection, et qui nécessite une intervention approfondie sur ce dernier, comme la modification des fichiers de configuration et la manipulation des règles de filtrage.

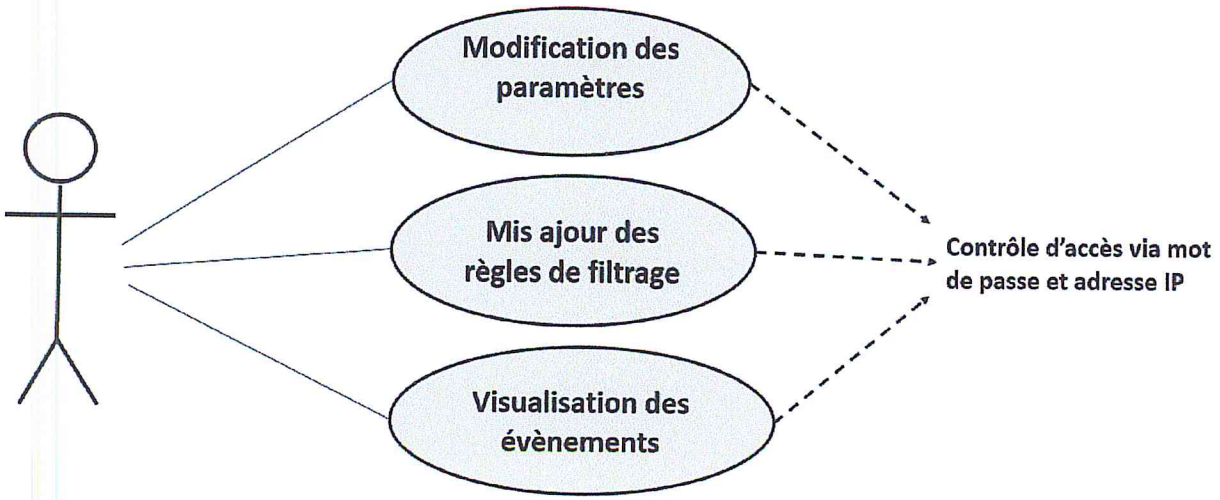


Figure 31 – Diagramme de cas d'utilisation du module implémenté.

2.3.1 Modification des paramètres :

L'un des fonctionnalités de cette interface d'administration du pare-feu web est de modifier la configuration de base de pare-feu :

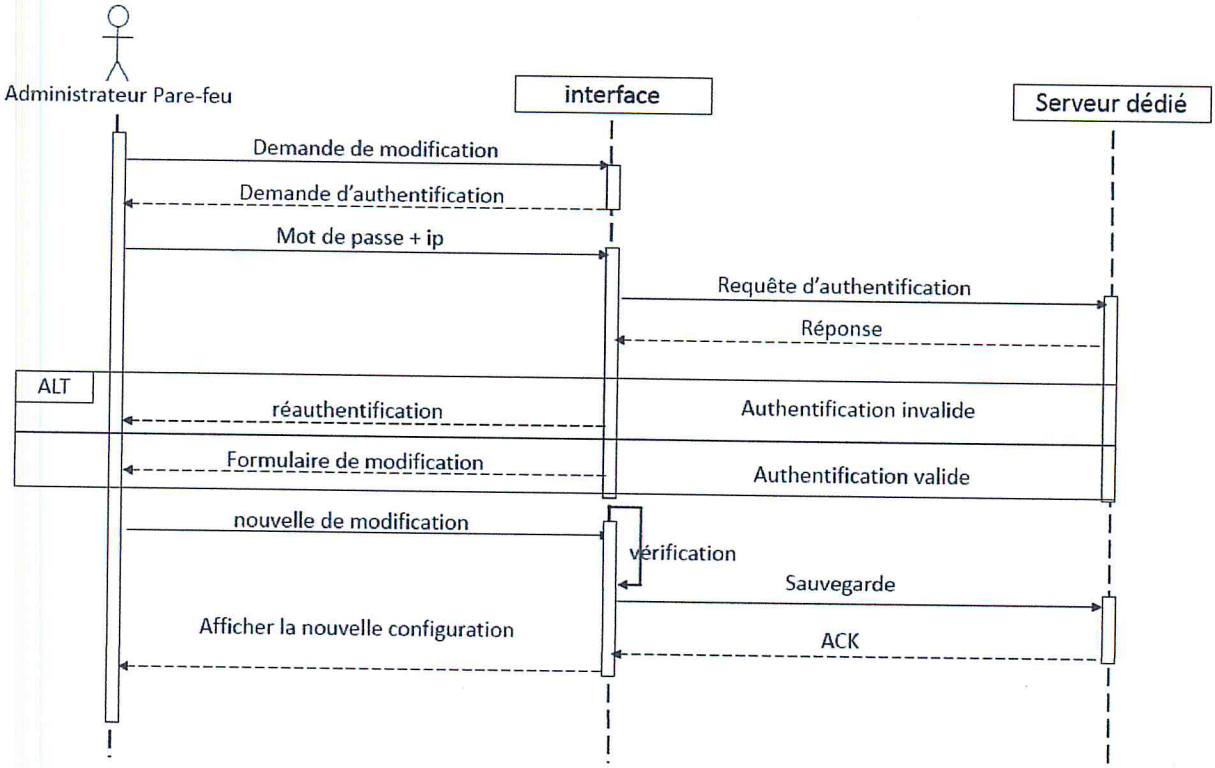


Figure 32 - Diagramme de séquence d'une nouvelle configuration de pare-feu.

2.3.2 Gestion des règles de filtrage :

Une autre fonctionnalité de cette interface de pare-feu web est de manipuler des règles de filtrage. On peut ajouter, modifier ou supprimer des règles.

Le comportement de ces fonctionnalités est ci-dessous :

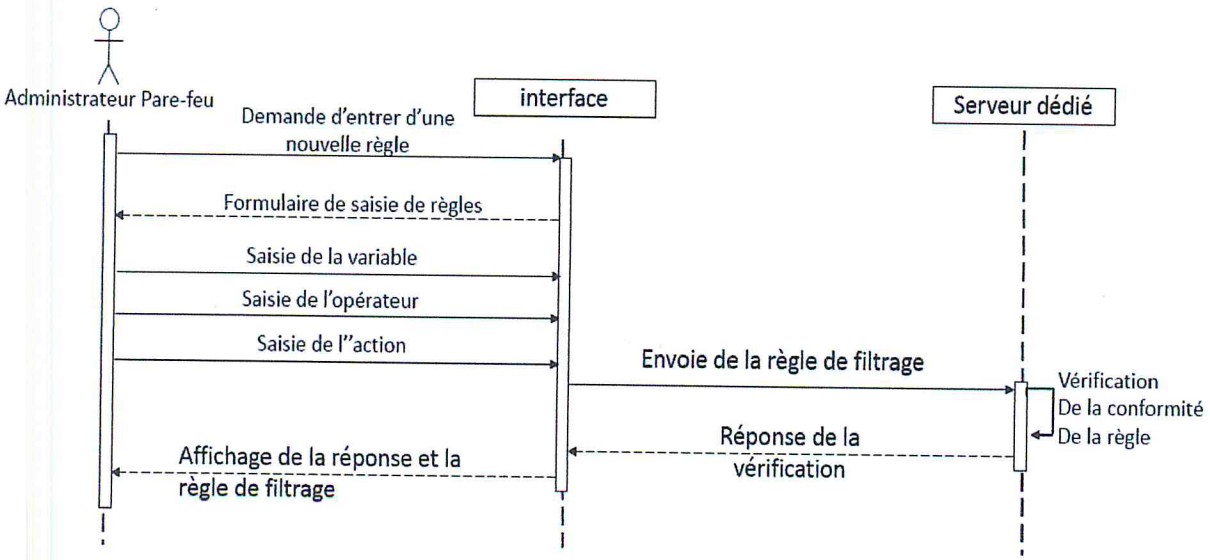


Figure 33 - Diagramme de séquence de l'ajout d''une nouvelle règle de filtrage de pare-feu.

Le comportement de la modification de règles ce fait presque de la même manière que l'ajout d'une règles :

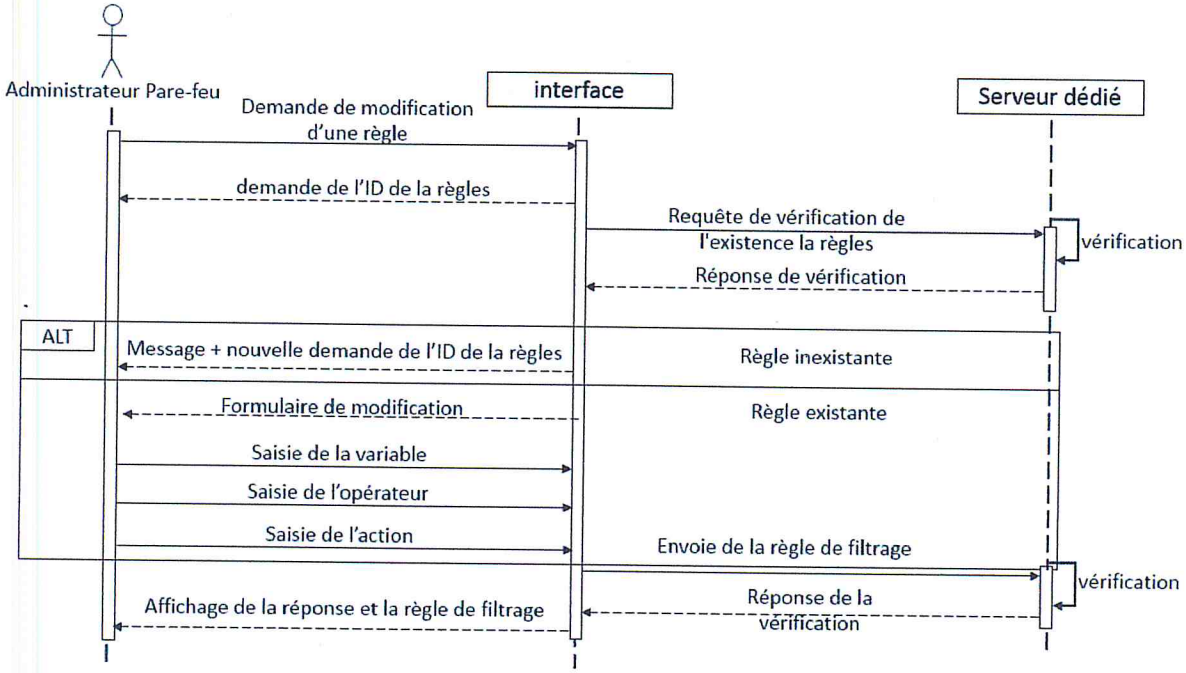


Figure 34 - Diagramme de séquence de la modification d'une règle de filtrage de pare-feu.

On peut aussi supprimer une règle de filtrage carrément si elle n'est plus utile pour nous, d'où le comportement suivant :

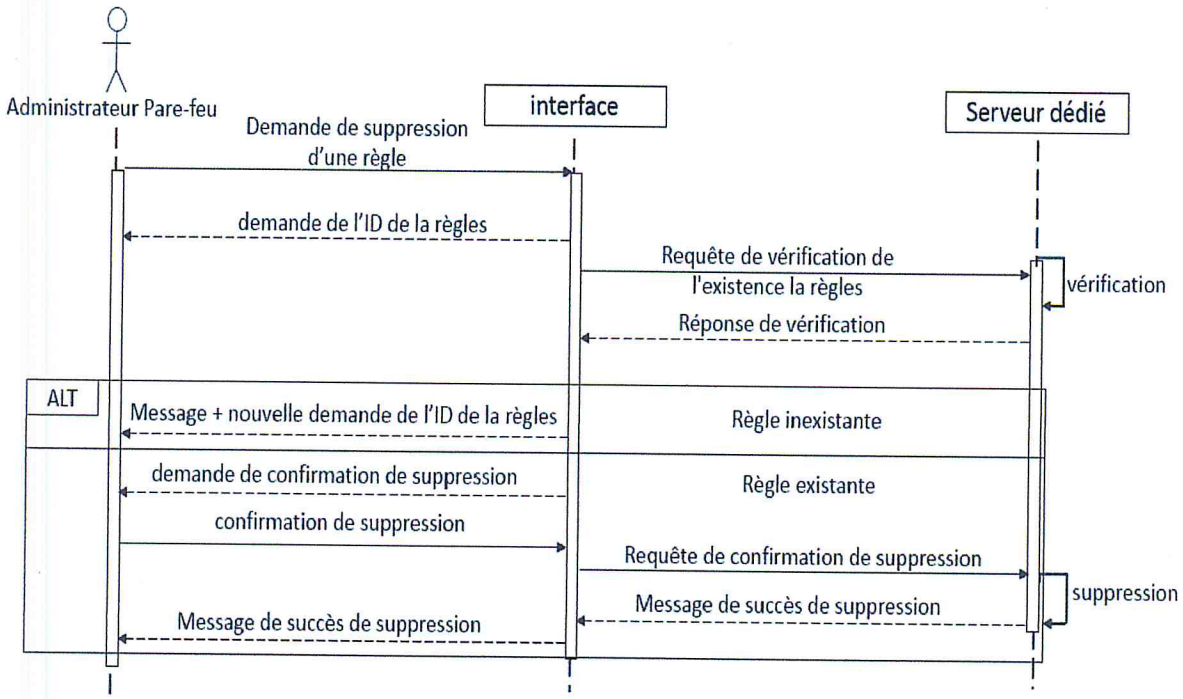


Figure 35 - Diagramme de séquence de la suppression d'une règle de filtrage de pare-feu.

2.3.3 Visualisation des évènements :

Cette fonctionnalité est essentielle elle permet le monitoring et une meilleure surveillance de notre système, vu que ce projet accorde de l'importance à l'ergonomie mais surtout à la pertinence de l'information ou une image moderne de contrôle de sécurité est espérée.

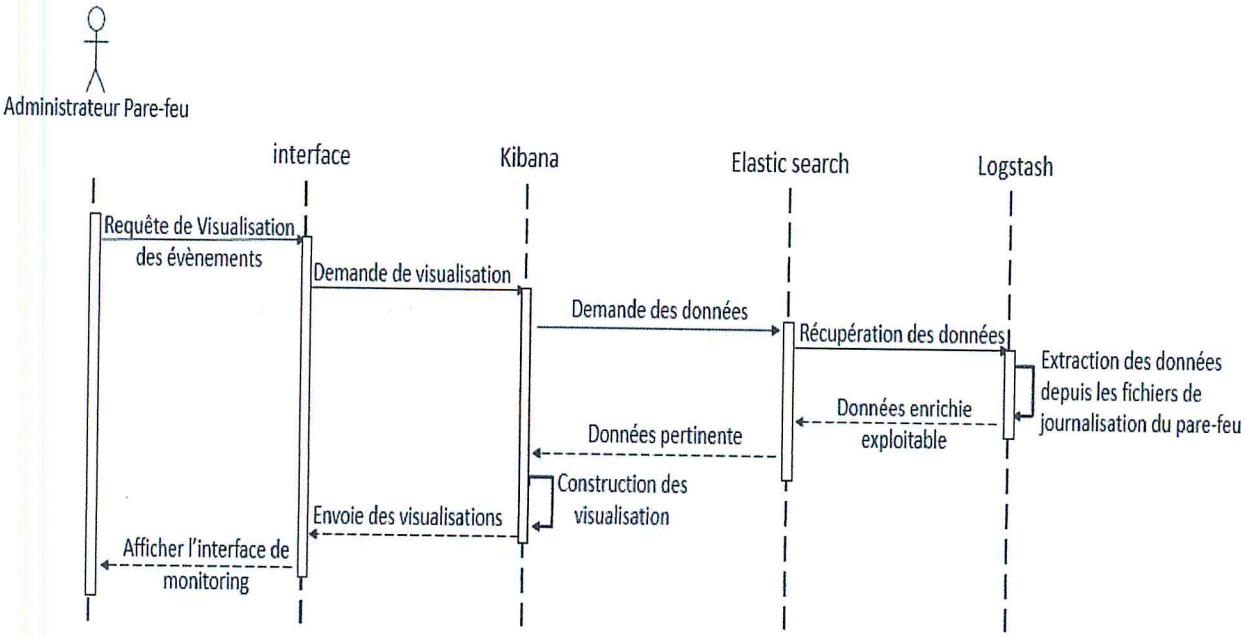


Figure 36 - Diagramme de séquence qui montre le comportement d'interface de monitoring.

3. Conclusion :

Dans ce chapitre nous avons présenté l'architecture du système mis en place, nous avons abordé comment renforcer la sécurité d'applications web via la couche indépendante « pare-feu web », nous avons présenté quelques parades pour atténué les attaques avec notre WAF, bien que le pare-feu web nécessite encore plus d'approfondissement dans les règles de filtrage lors d'une utilisation réelle pour approuver les résultats. Ainsi on a mis en place un module d'administration et de monitoring du pare-feu déployé, qui est une image sur les incidents qui se produise, ce qui nous offre une connaissance approfondie sur le trafic entrant, ceci servira à améliorer et adapter la politique de sécurité à travers le temps.

Toutes les parties qui construire notre plateforme sont modélisé et détaillé dans ce chapitre. Dans le prochain chapitre nous allons exprimer l'étape de la réalisation.

2.1.1 Atténuation d'attaques via le WAF

Puisque la mission de ce nœud est une mission clef dans le système on va présenter une section pour une stratégie de mitigation d'attaques avec des règles de filtrage.

Évidemment certaines failles doivent être prises en compte au niveau de l'application, bien sûr on n'ouvre pas la porte aux bandits puis on se demande pourquoi on se fait voler. Un WAF est censé remédier à ces vulnérabilités étant le pont par lequel tout passe.

Présenter toutes les vulnérabilités n'est vraiment pas aisé dans notre contexte d'étude, car chacune des attaques nécessite en elle-même une recherche approfondie de ces usages théoriques et pratiques. C'est pourquoi nous expliquerons en bref les grandes lignes en tenant exemple de certaines de ces attaques et des approches de défense munies par le pare-feu. Les règles de politiques de nos configurations standards tirent profit de plusieurs signatures génériques comme OWASP ModSecurity Core Rule Set (CRS).

Bloqué les requêtes proxy

Les requêtes routées via les proxys peuvent être problème pour certains sites. Les utilisateurs peuvent se cacher derrière l'anonymat perçu d'un proxy serveur et lancer n'importe quoi, vous pouvez donc vouloir bloquer les requêtes soumises si vous trouvent qu'ils causent des problèmes sur votre site.

Une façon de le faire, c'est de vérifier la présence de l'en-tête X-Forwarded-For dans le requête HTTP. Si cet en-tête existe, cela signifie que la requête a été effectuée par un proxy serveur pour le compte de l'utilisateur réel.

Cette règle détecte et bloque les requêtes des serveurs proxy qui utilisent l'En-tête X-Forwarded-For:

```
SecRule &REQUEST_HEADERS: X-Forwarded-For "@gt 0" deny
```

La règle utilise l'opérateur '&' pour énuméré le nombre d'en-têtes de requête qui présentes le nom X-Forwarded-For. Si ce nombre est supérieur à zéro, cela signifie qu'un en-tête est présent et la requête est bloquée.

Les injections

Les attaques par injection se produisent lorsqu'une donnée non contrôlée est envoyée à un interpréteur dans le cadre d'une commande ou d'une requête afin d'exécuter des actions imprévues ou d'accéder à des données non autorisées.

les différentes attaques par injection reposent principalement sur le même principe qui est l'utilisation de mots clefs et de caractères spécifiques qui permettent par exemple de mettre en commentaire des portions de code et d'insérer du code frauduleux. Il est cependant rare que l'application ait besoin d'accepter tous ses éléments, comme :

```
` ' " \ * / = + ! & < > ~ # { [ ( ) ] } | ^ $ @ . , : ;
```

Anticiper ces attaques consiste donc à bien empêcher les requêtes porteuses de caractère spécial ou mot clé.

Cette règle détecte et bloque les requêtes qui contiennent des caractères qui peut s'agir d'une injection:

```
SecRule ARGS_GET "@contains (caractere dangeureu)" "deny,msg:'SQL Injection'"
```

Prévenir des attaques XSS

La mesure la plus importante que vous pouvez prendre pour éviter les attaques XSS est de s'assurer que toutes les données fournies par l'utilisateur qui sont affichées dans vos pages Web sont correctement désinfectées. Cela signifie que le remplacement des caractères potentiellement dangereux, tels que les parenthèses incluses (<et>) avec leurs versions codées de l'entité HTML correspondantes, dans ce cas < Et &gt.

Voici une liste de caractères que vous devez encoder lorsqu'ils sont présents dans des données fournies par l'utilisateur pour être incluses dans les pages Web :

caractères	HTML-encode
<	&lt ;
>	&gt ;
(	&#40 ;
)	&#41 ;
#	&#35 ;
&	&amp ;
''	&quot ;
'	&#39 ;

```
SecRule REQUEST_BODY "@detectXSS" "id:12345,log,deny"
```

Prévenir les attaques de force brute

Les attaques de force brutale permettent à un attaquant d'accéder à une ressource en essayant à plusieurs reprises de deviner des noms d'utilisateur, des mots de passe, des adresses de courrier électronique ou d'autres ressources critiques, elles peuvent être très efficaces si aucune protection n'est mise en place.

Une bonne façon de se défendre contre les attaques de force brute est de permettre un certain nombre de tentatives de connexion, disent trois, et après cela, commencer à retarder ou à bloquer d'autres tentatives. Voyons comment nous pouvons utiliser ModSecurity pour y parvenir.

Si votre page de connexion login est située à votresite.com/login, alors les règles suivantes suivront le nombre de tentatives de connexion des utilisateurs :

```
Bloquer les tentatives de connexion après 3 tentatives échouées
<LocationMatch ^/login>
# initialiser la collection IP avec l'adresse IP de l'utilisateur
SecAction "initcol:ip=%{REMOTE_ADDR},pass,nolog"

# Détecter les tentatives de connexion échouées
SecRule RESPONSE_BODY "Username does not exist" "phase:4,pass,setvar:
ip.failed_logins=+1,expirevar:ip.failed_logins=60"

# Bloquer les tentatives de connexion ultérieures
SecRule IP:FAILED_LOGINS "@gt 3" deny
</Location>
```

Les règles initialise la collection IP et augmenter les ip.failed_logins après chaque tentative de connexion échouée. Une fois plus de trois connexions échouées sont détectés, les prochaines tentatives sont bloquées. L'action **expirevar** est utilisée pour réinitialiser le nombre de tentatives de connexion échouées à zéro après 60 secondes, de sorte que le blocage sera en vigueur pendant une durée maximale de 60 secondes.

2.2 Nœud d'acquisition de données

Gérer les évènements générés par le pare-feu est la mission principale de ce nœud, dans cette section le but est de chercher, analyser, et structurer les données des fichiers log du pare-feu, pour qu'elles deviennent exploitables par des outils de programmation. A travers cette partie du système nous cherchons à récolter des données pertinentes et construire une base de données centralisé qui va être par la suite utilisé pour surveiller les évènements qui se produise dans une interface de monitoring, et dans le même contexte nous cherchons à développer une base de connaissance qui nous aide à améliorer la politique de sécurité.

On à exploiter la suite ElasticStack (Logstash, Elasticsearch, et Kibana) version 5.0 pour les échanges de données entre le pare-feu web et l'IHM, c'est une suite logicielle pour le traitement de données en temps réel. Nous avons choisi cette solution pour ça performance en matière de recherche et d'analyse de données structurées et non structurées, elle est beaucoup utilisé lors du passage à l'échelle (big data) pour son moteur de recherche puissant. Le rôle de ce nœud se résume dans ces étapes :

2.2.1 Logstash (pipeline)

Le rôle de ce module est de récupérer les données depuis une source où ils sont éparpillés puis procéder à l'analyse de ces données afin de les filtrer et structurer puis les transporter vers un module de stockage. D'où nous avons tiré profit de l'outil (logstash) pour pouvoir analyser et filtrer les logs de notre pare-feu web afin d'obtenir les données voulues. Nous utilisons logstash comme un pipeline entre la source de la donnée qui est le fichier de journalisation et le module de stockage (elasticsearch), il nous permet d'analyser, chercher, et extraire des données pertinentes de puis les fichiers logs. Le schéma de fonctionnement de logstash :

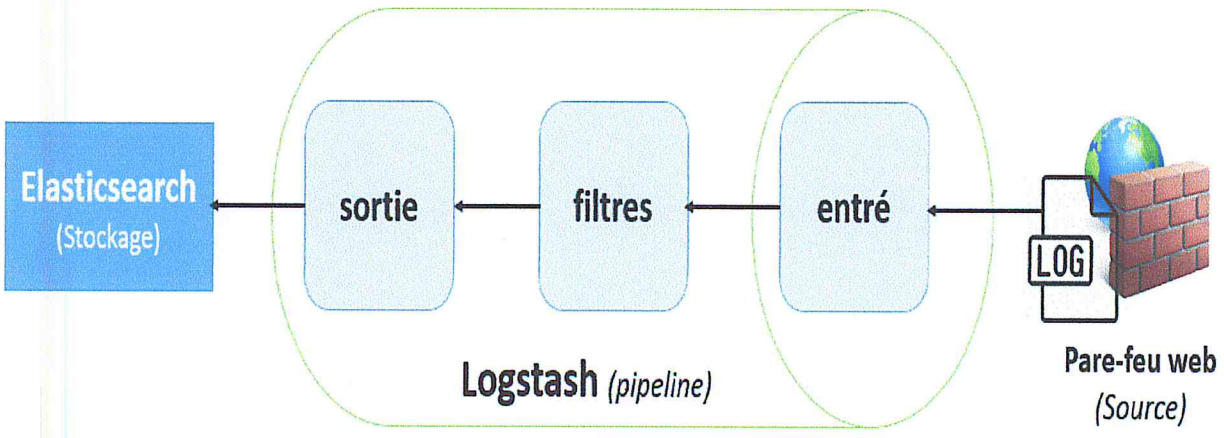


Figure 24 - fonctionnement du pipeline (logstash) dans notre architecture

Les traitements dans le pipeline (Logstash) sont composés en trois parties (entrée, filtre, sortie):

Entrée :

Au cours de cette partie on doit spécifier l'emplacement de l'ingestion des données. Dans notre cas c'est les fichiers logs de pare-feu web. Dans un second traitement on doit gérer correctement les logs qui sont un texte multi-lignes, on doit savoir comment dire quelles lignes font partie d'un seul événement. Il est évident que chaque début d'évènement commence avec cette étiquette - - 5759e83f - A - - qui est le marqueur d'entrée de chaque nouveau log d'évènement, où la partie souligné en rouge est variable, et inversement le reste est fixe. Pour déterminer les lignes de logs qui sont du même évènement nous avons mis en place une expression régulière qui identifie cette suite de caractères.

L'automate suivant montre le fonctionnement de l'analyse syntaxique qui identifier la chaine de caractères qui affirme un nouvel évènement :

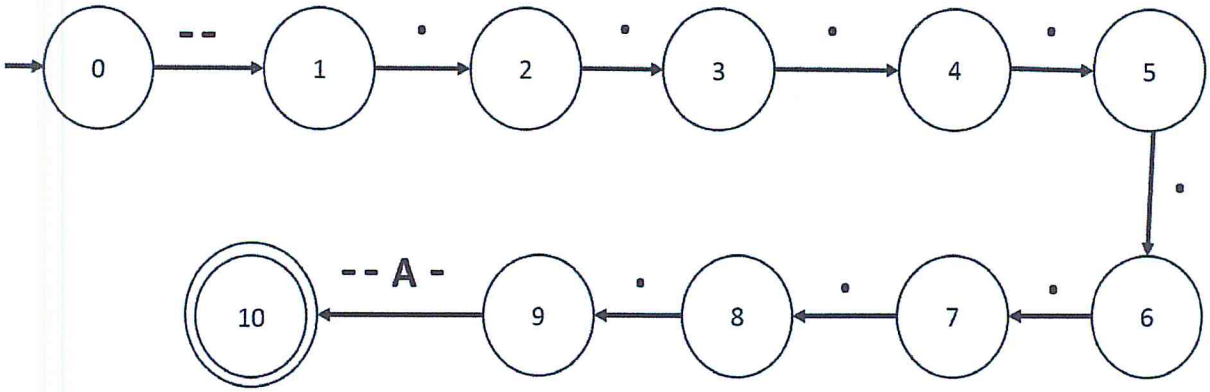


Figure 25 – automate qui détermine un nouvel évènement

Cela consiste à analyser les logs en constituants une chaîne composés des unités lexicales suivantes :

A : Constante alphabétique

- : Constante caractère spéciale

. : Variable qui signifie tout caractère alphanumérique

L'automate ci-dessus doit vérifier l'existence de la suite de caractère qui correspond à l'expression régulière de logstash : ^ - - . . . . . - A - - , le symbole ^ signifie le début du texte ou de la ligne. Si la vérification est confirmé ça veut dire que les lignes suivante appartiennent au même évènement jusqu'à la réapparition d'une nouvelle expression similaire.

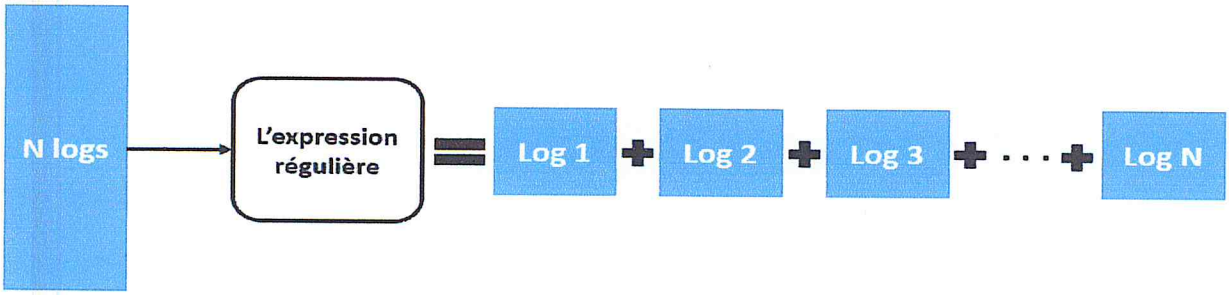


Figure 26 –Rôle de l'expression régulière

2. Filtre :

Le but de cette partie est d'analyser les logs pour filtrer les données bruités, et extraire les données pertinentes. Pour extraire ces données pertinentes nous avons adapté un modèle de correspondance (pattern matching) conforme au format du log d'un évènement d'attaque, et ainsi filtrer et garder que les données pertinentes que nous voulons exploiter. Cette étape permet la structuration des données éparpillées dans les fichiers logs.

La figure 28 est un organigramme qui résume l'étape de filtration du fichier de journalisation :

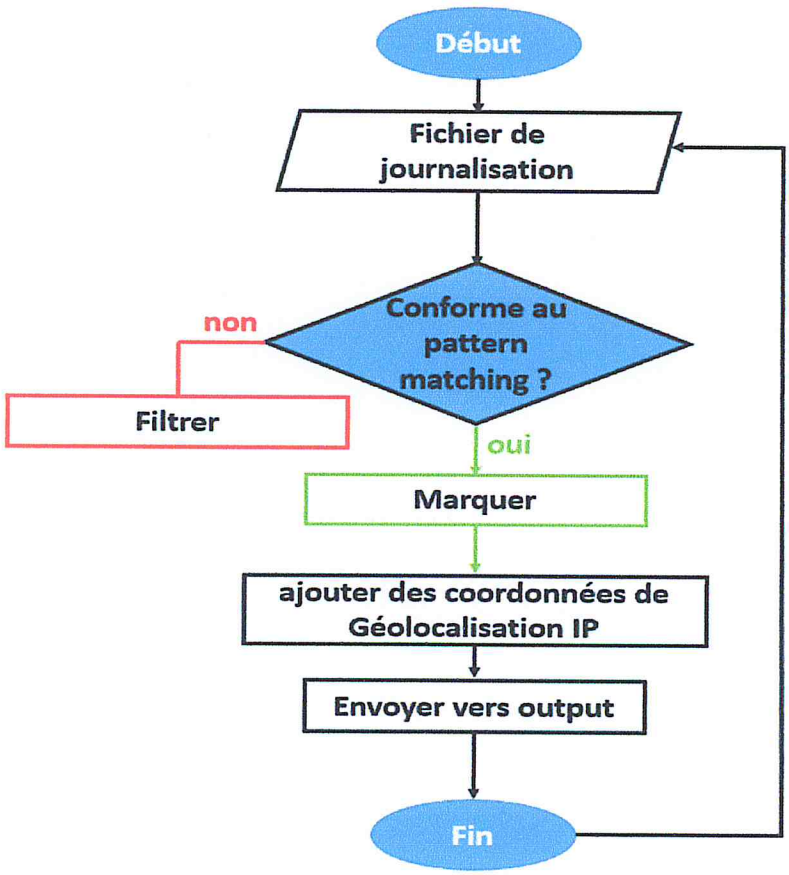


Figure 27 – organigramme du filtre des logs.

Extraction des donnés via Pattern matching :

Dans le filtre nous avons mis en place un pattern matching qui permet de trouver les données des logs souhaité. Dans ce sens nous avons utilisé **Grok** qui est une fonction de logstash qui permet de correspondre les logs avec des patterns et de les transformer en données structuré consultable.

Grok combine les patterns de texte prédéfinis avec le format qui correspond aux journaux.

La syntaxe d'un pattern **Grok** est %{SYNTAX : SEMANTIC}

SYNTAX : est le nom du pattern prédéfinis qui correspond à notre texte. Par exemple, 3.44 sera apparié par le modèle NUMBER et 55.3.244.1 sera le modèle IP. La syntaxe est la façon dont vous faites correspondre.

SEMANTIC : est l'identifiant (tague / marque) que nous donnons au morceau de texte correspondant. Par exemple, 3.44 pourrait être la durée d'un événement, donc vous pouvez l'appeler simplement la durée. En outre, une chaîne 55.3.244.1 pourrait identifier le client en faisant une demande.

Avec ce principe de syntaxe et sémantique, nous pouvons extraire des champs utiles à partir d'un journal, nous expliquons ce mécanisme avec un échantillon dans l'exemple suivant.

Exemple : 55.3.244.1 GET /index.html 15824

Pour cela le pattern pourrait être :

%{IP:client} %{WORD: méthode } %{URIPATHPARAM: requête } %{NUMBER:bytes}

Un exemple dans logstash:

```
filter {
grok {match =>
{"inputs" => "%{IP : client} %{WORD : méthode} %{URIPATHPARAM : requête} %{NUMBER:bytes} }
 }
}
```

Après le filtre grok, l'événement aura comme résultat les champs ci-dessous :

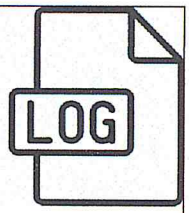
-client : 55.3.244.1

-méthode : GET

-requête : /index.html

-bytes : 15824.

Les parties colorées en jaune c'est les champs que nous souhaitons extraire pour les exploiter, le reste du log sera filtrer.




```
--5759e83f-A--
[27/Mar/2017:14:22:32 +0000] dqIu7V5MziQAAEpPAWwAAAAE 94.76.206.36 38037 94.76.206.36 80
--7a19264b-B--
GET /a/a2/passage.php?login=admin%27+or+%271%27%3D%271%27%23&password=azzertt HTTP/1.1
Host: 127.0.0.1
User-Agent: nikto.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1/a/a2/
Connection: keep-alive
--5759e83f-F--
HTTP/1.1 403 Forbidden
Content-Length: 275
--7a19264b-E--
<html><head>
<title>403 Forbidden</title>
</head><body>
<p>You don't have permission to access /a/a2/passage.php on this server.</p></body></html>
--7a19264b-H—
Message: Access denied with code 403 (phase 2). Pattern match
"(?i:([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)\\b([\\d\\w]++)([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*
?)(?:(?:=|<=>|r?like|sounds\\s+like|regexp)([\\s'\"`\xc2\xb4\xe2\x80\x99\xe2\x80\x98\\(\\)]*?)\\2\\b|(?:!=|<=|>=|<>|<|
>|\\^|is\\s+not ..." at ARGS:login. [file "/usr/share/modsecurity-crs/activated_rules/modsecurity_crs_41_sql_injection_
attacks.conf"] [line "77"] [id "950901"] [rev "2"] [msg "SQL Injection Attack: SQL Tautology Detected."] [data "Matched
Data: '1'='1 found within ARGS:login: admin' or '1'='1'#"] [severity "CRITICAL"] [ver "OWASP_CRS/ 2.2.8"] [maturity "9"]
[tag "WASCTC/ WASC-19"] [tag "OWASP_TOP_10/A1"] ..... .....
```

Figure 28 - les parties de log souhaité

Nous avons choisie de extraire uniquement certain parties de logs pour deux raisons, sois car les parties extraites peut déterminer les données filtré, ou bien parce que c'est des informations sans bénéfices dans notre contexte d'étude (exp : version serveur, message d'erreurs … etc).

Sortie :

Après avoir filtré les données, logstash envoie les données extraites vers le module de stockage (elasticsearch) qui est chargé de les stockée d'une manière fiable et bien structurer.

2.2.2 Elasticsearch (module de stockage)

Nous comptons utiliser cet Elasticsearch comme un nœud de stockage qui permet d'indexer et stocker ces données de manière centralisée pour pouvoir les interroger en toutes circonstances. Cet outil est un moteur de recherche et d'analyse de données open-source hautement évolutif, il nous permet de rechercher de gros volumes de données rapidement et en temps quasi réel, il est utilisé comme moteur / technologie sous-jacent qui gère les exigences de recherche complexes.

Le but de ce stockage est de construire une base de connaissance exploitable qui va être affiché sur une interface de monitoring ainsi avoir une bonne image qui nous aide optimiser notre politique de sécurité. Le rôle de cet outil est illustré dans ce petit schéma :

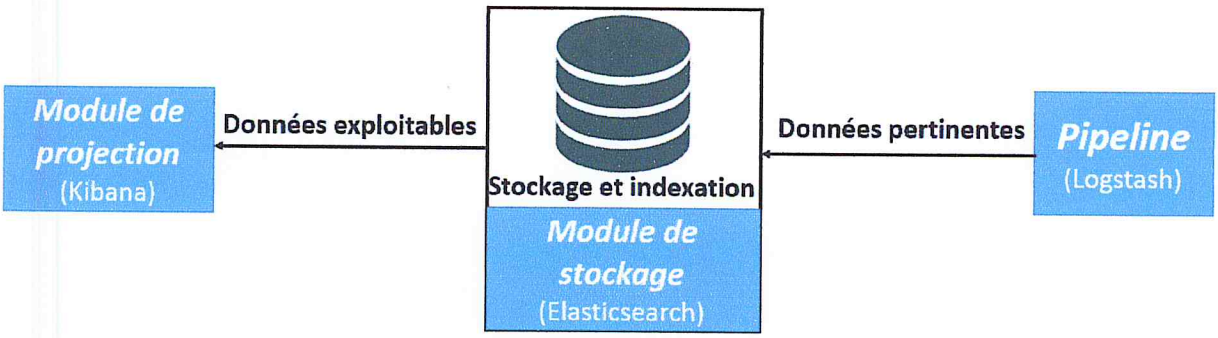


Figure 29 – Rôle du module de stockage (elasticsearch) dans notre plateforme

La table suivante montre l'ensemble des données pertinentes extraite de puis le fichier de journalisation du pare-feu web, ces données seront structuré, stocké et puis interroger.

***Tableau 6** – table de données pertinente extraites*

Données pertinentes	
- Id	- geoip.country_name
- timestamp	- geoip.continent_code
- User Agent	- geoip.city_name
- IP source	- geoip.ip
- port source	- geoip.location
- IP destination	- geoip.timezone
- destination Port	
- HOST	
- REFERER	
- Version http	
- méthode Http	
- requête	
- type d'attaque	
- gravité	
- rule id	
- rule data	
- rule file	

2.2.3 Kibana (module de projection)

Kibana est un outil qui permet de consulter Elasticsearch et d'explorer nos données stocké, on peut facilement projeter une analyse de nos données dans une variété de tableaux, courbes, figures ou cartes. Ces visualisations qui interrogeant les données qu'on a préalablement structuré et stocké, pour a la fin pouvoir les intégré dans notre interface de monitoring.

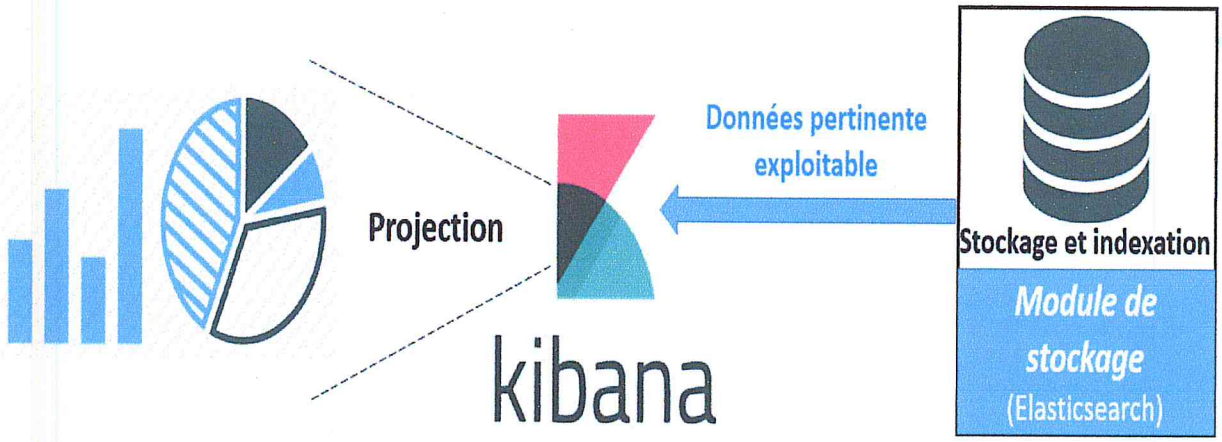


Figure 30 – Rôle de module de projection (Kibana) dans notre plateforme

2.3 L'interface homme machine

L'interface graphique d'administration et de monitoring doit supporter toute option de configuration fournie par le nœud de protection, et qui nécessite une intervention approfondie sur ce dernier, comme la modification des fichiers de configuration et la manipulation des règles de filtrage.

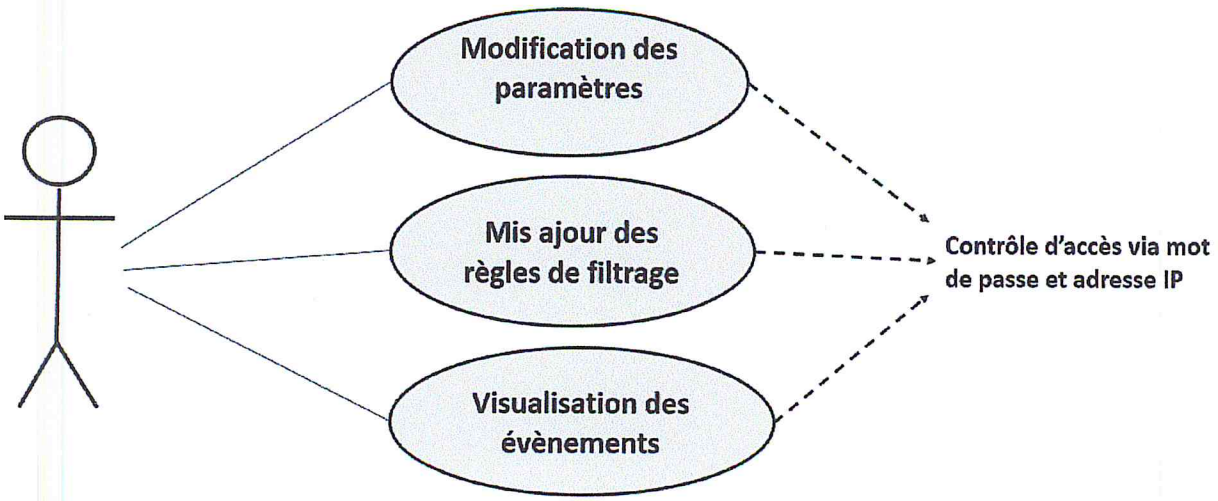


Figure 31 – Diagramme de cas d'utilisation du module implémenté.

2.3.1 Modification des paramètres :

L'un des fonctionnalités de cette interface d'administration du pare-feu web est de modifier la configuration de base de pare-feu :

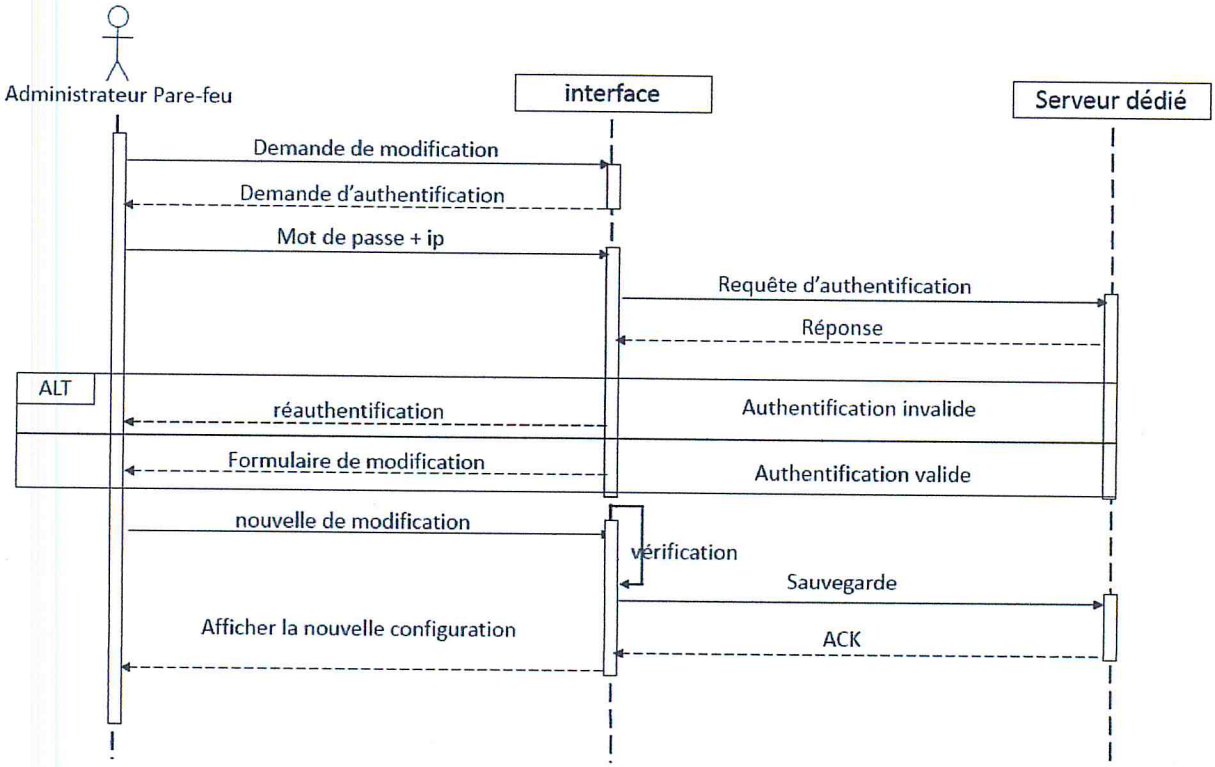


Figure 32 - Diagramme de séquence d'une nouvelle configuration de pare-feu.

2.3.2 Gestion des règles de filtrage :

Une autre fonctionnalité de cette interface de pare-feu web est de manipuler des règles de filtrage. On peut ajouter, modifier ou supprimer des règles.

Le comportement de ces fonctionnalités est ci-dessous :

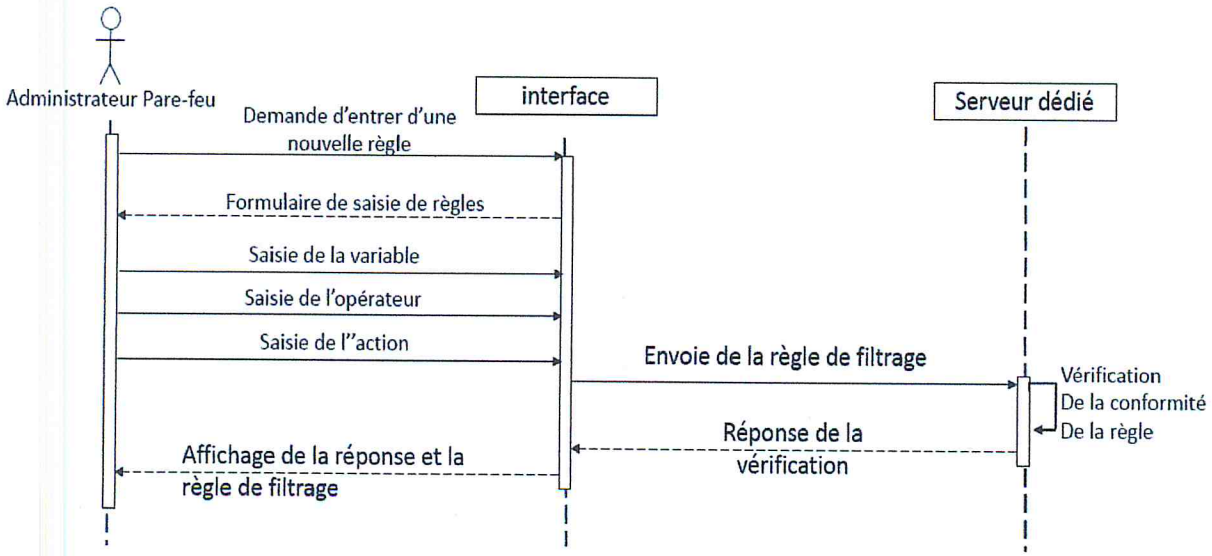


Figure 33 - Diagramme de séquence de l'ajout d''une nouvelle règle de filtrage de pare-feu.

Le comportement de la modification de règles ce fait presque de la même manière que l'ajout d'une règles :

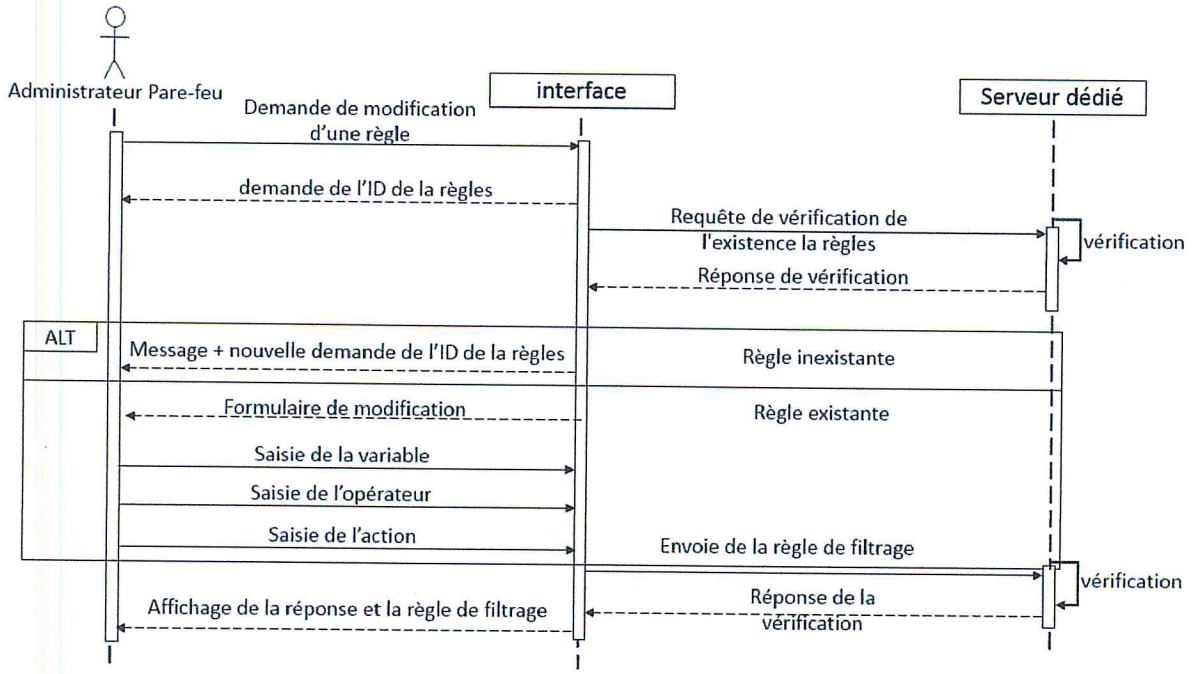


Figure 34 - Diagramme de séquence de la modification d'une règle de filtrage de pare-feu.

On peut aussi supprimer une règle de filtrage carrément si elle n'est plus utile pour nous, d'où le comportement suivant :

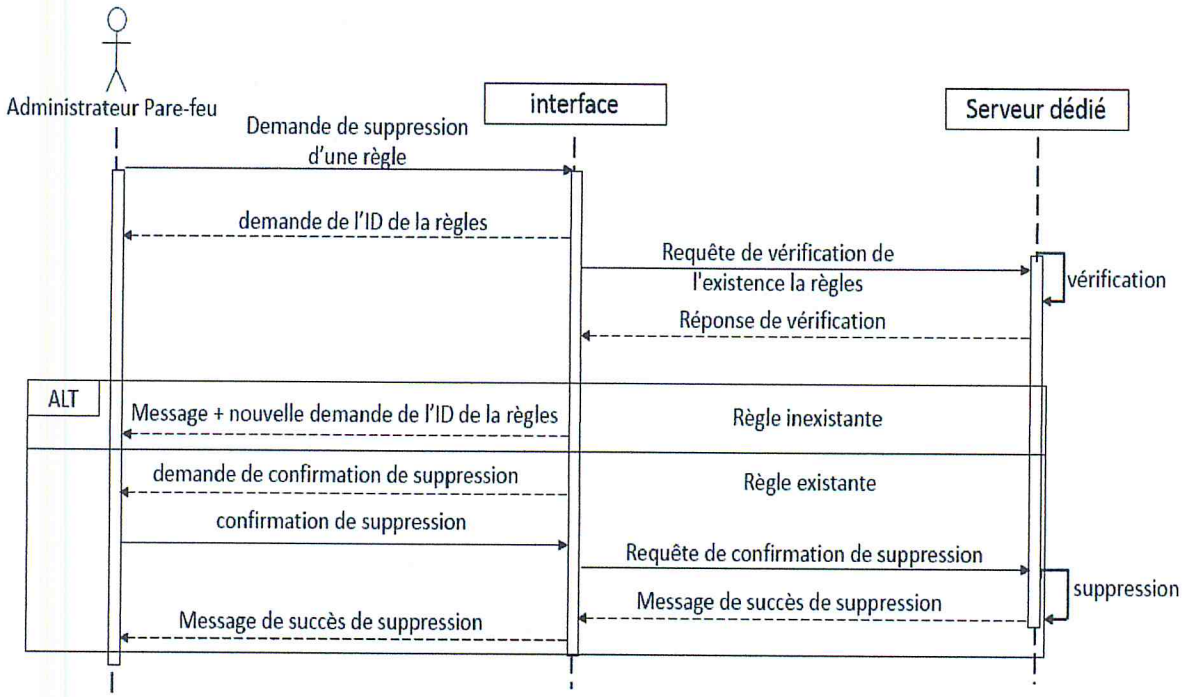


Figure 35 - Diagramme de séquence de la suppression d'une règle de filtrage de pare-feu.

2.3.3 Visualisation des évènements :

Cette fonctionnalité est essentielle elle permet le monitoring et une meilleure surveillance de notre système, vu que ce projet accorde de l'importance à l'ergonomie mais surtout à la pertinence de l'information ou une image moderne de contrôle de sécurité est espérée.

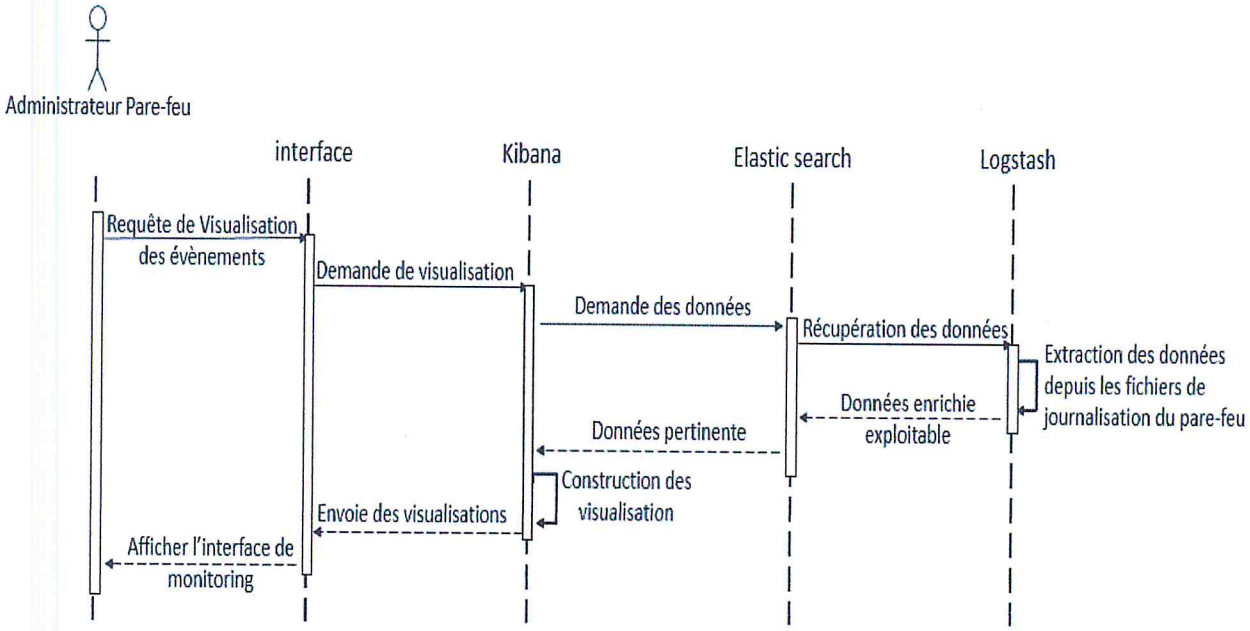


Figure 36 - Diagramme de séquence qui montre le comportement d'interface de monitoring.

3. Conclusion :

Dans ce chapitre nous avons présenté l'architecture du système mis en place, nous avons abordé comment renforcer la sécurité d'applications web via la couche indépendante « pare-feu web », nous avons présenté quelques parades pour atténué les attaques avec notre WAF, bien que le pare-feu web nécessite encore plus d'approfondissement dans les règles de filtrage lors d'une utilisation réelle pour approuver les résultats. Ainsi on a mis en place un module d'administration et de monitoring du pare-feu déployé, qui est une image sur les incidents qui se produise, ce qui nous offre une connaissance approfondie sur le trafic entrant, ceci servira à améliorer et adapter la politique de sécurité à travers le temps.

Toutes les parties qui construire notre plateforme sont modélisé et détaillé dans ce chapitre. Dans le prochain chapitre nous allons exprimer l'étape de la réalisation.

Chapitre 5

Implémentation

1. Introduction

Dans ce chapitre, nous allons présenter l'environnement de développement de notre système en précisant les outils et les langages de programmation qu'on a choisie pour le réaliser. Puis on va présenter un aperçue des fonctionnalités de l'implémentation proposées dans le chapitre précédent, ainsi que les principales interfaces qui la compose.

2. Environnement de travail

Nous allons commencer par la description de l'environnement de travail et les outils choisie pour la réalisation de notre système.

2.1 La virtualisation

La virtualisation des ressources nous permet de bénéficier d'une optimisation de valeur, actuellement nous utilisons des conteneurs de virtualisation basés sur VMware Workstation Pro v12 qui permettent d'isoler nos systèmes d'exploitation.

2.2 Pare-feu web

Dans ce contexte nous avons exploité modsecurity installer sur une machine virtuelle Linux Ubuntu 16.04 contenant un serveur Apache, le pare-feu en architecture reverse-proxy comme il est montré dans le chapitre précédent de l'architecture des systèmes.

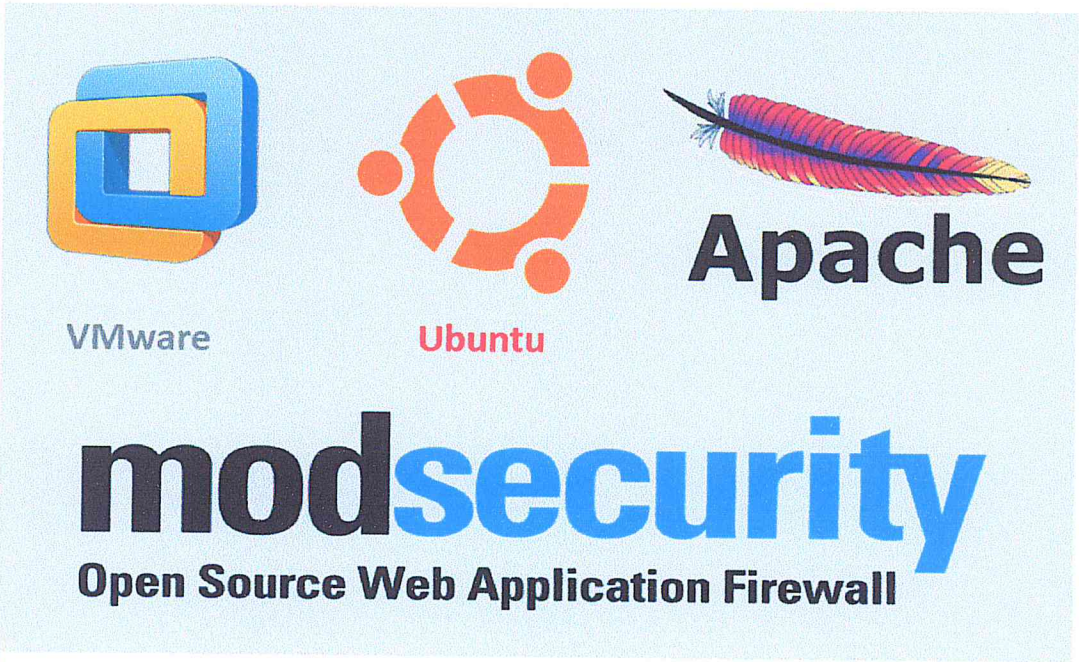


Figure 37 - Les outils utilisés pour le déploiement du pare-feu web

2.3 Les outils de développement de l'application

Nous avons utilisé plusieurs langages et outils pour réaliser notre application, ce qu'il nous a permis de mettre en place un système dynamique et interactif, ces langages sont dans la figure suivante.

Figure 38 - langage et outils utilisés pour le développement de l'application

On a exploité des outils, comme :

PHP 5 (Hypertext Preprocessor) **:** Est un langage de programmation pour les applications destiné à être intégré dans des pages HTML. Principalement dédié à la production de pages HTML générées dynamiquement.

HTML 5 (HyperText Mark-up Language) **:** C'est un langage de balisage moderne dans lequel sont écrites les pages du web. Il contient également un ensemble éléments et de formulaire qui devraient pour le développement d'applications web.

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

Pour *Logstash, Elasticsearch et Kibana* on a vu un extrait sur ces outils et leurs rôles dans le chapitre précèdent, c'est la suite utilisé pour manipuler les données.

3. L'interface implémentée

Notre projet accorde une grande importance à l'ergonomie, à la convivialité et à une image moderne, ceci offre un confort lors de l'utilisation, ainsi ce critère peut faire la différence dans contrôle adéquat du système.

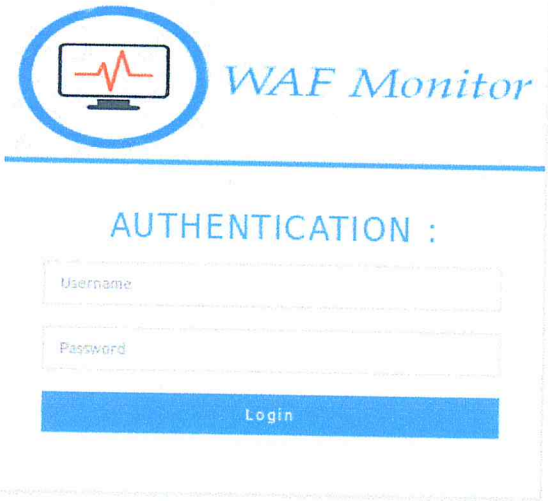


Figure 39 – interface d'authentification d'accès au module implémenté

On va voir maintenant un ensemble de capture d'écrans sur les principaux points de l'application :

3.1 L'interface de configuration du pare-feu

Cette capture d'écran est un extrait de la page qui permit de gérer la configuration principale de notre pare-feu :

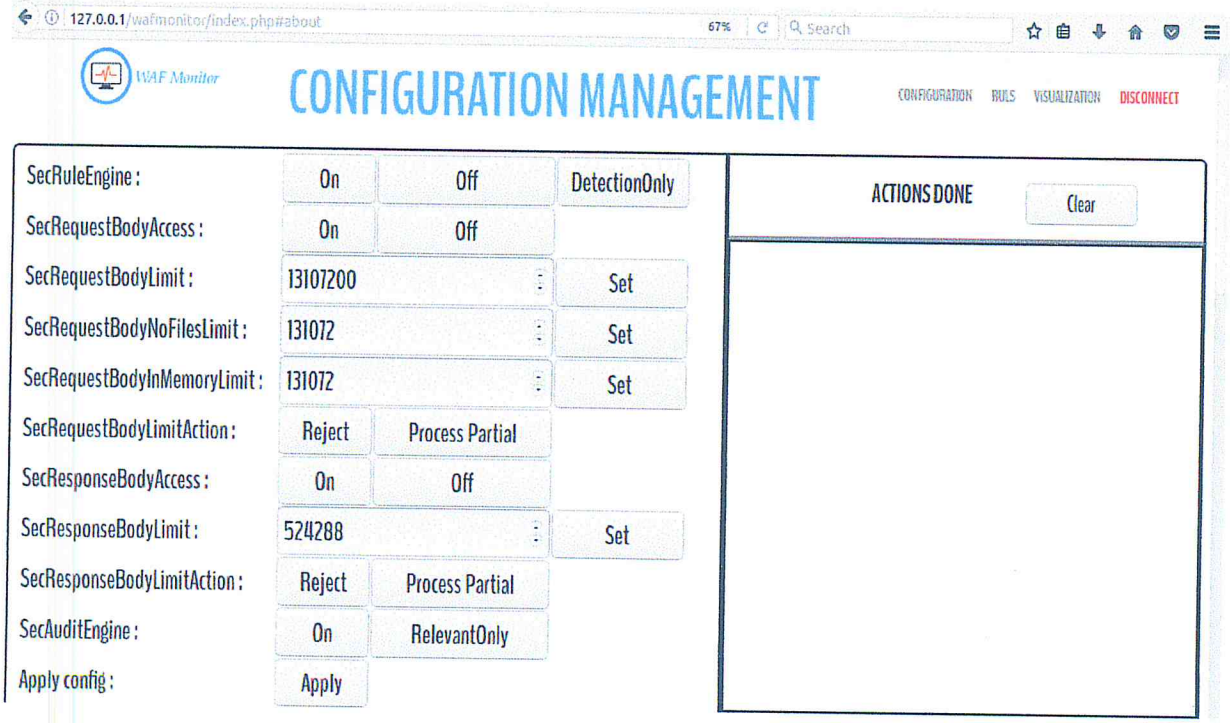


Figure 40 - interface de modification de configuration

L'interface présente dans la figure 40 permet de manipuler la configuration dans les points suivants :

1. Activer ou désactiver le pare-feu.
2. Activer ou désactiver l'analyse des requêtes http
3. Activer ou désactiver l'analyse des réponses http
4. Limiter la taille des requêtes http
5. Limiter la taille des réponses http
6. Limiter la taille des requêtes http qui ne contient pas de fichiers
7. Configurer l'action au cas d'abus de taille
8. Activer ou désactiver la journalisation des évènements

3.2 L'interface de gestion des règles de filtrage

Dans cette interface l'utilisateur peut ajouter modifier ou bien supprimer une règles du pare-feu web, lors de la l'ajout ou la modification quand l'utilisateur valide sa règle de filtrage le système vérifier la syntaxe de règle si elle est conforme, sinon un message d'erreur sera afficher.

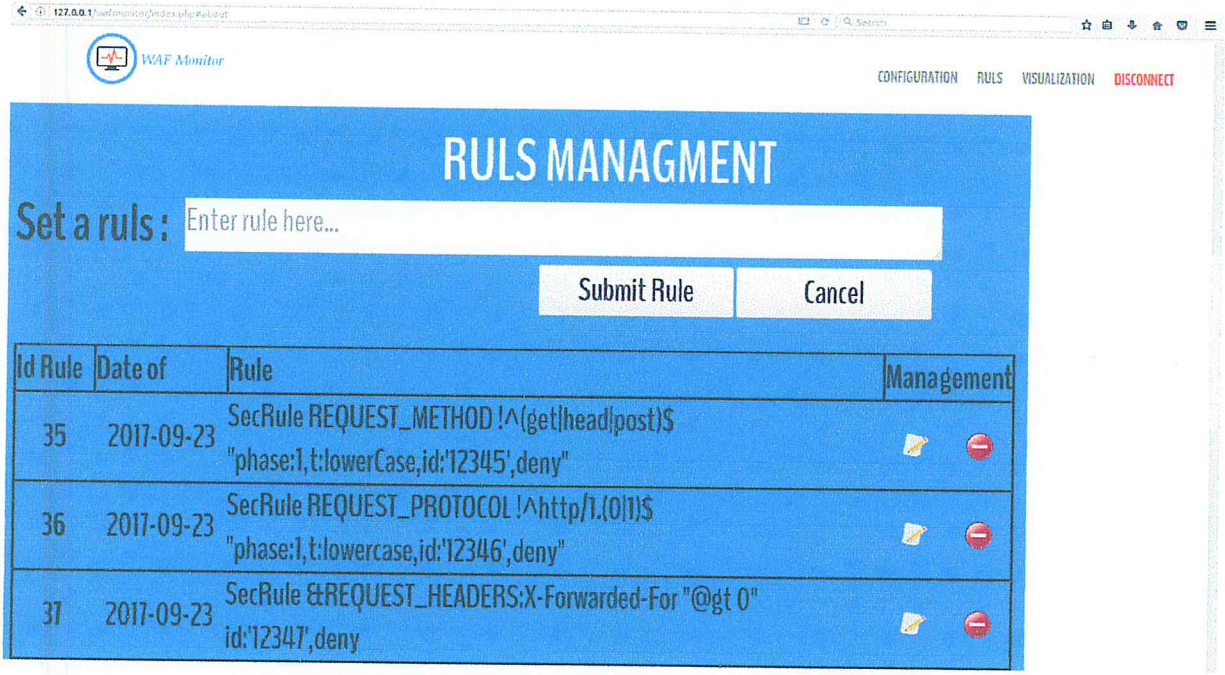


Figure 41 - interface de gestion des règles de filtrage

3.3 Aperçue de l'interface de monitoring

On peut dire que c'est l'interface principale de l'application elle permet de surveiller notre pare-feu, tous les tentatives d'attaque seront reporter sur cette interface. C'est une table de monitoring qui ou l'utilisateur peut trouver plusieurs visualisations significative.

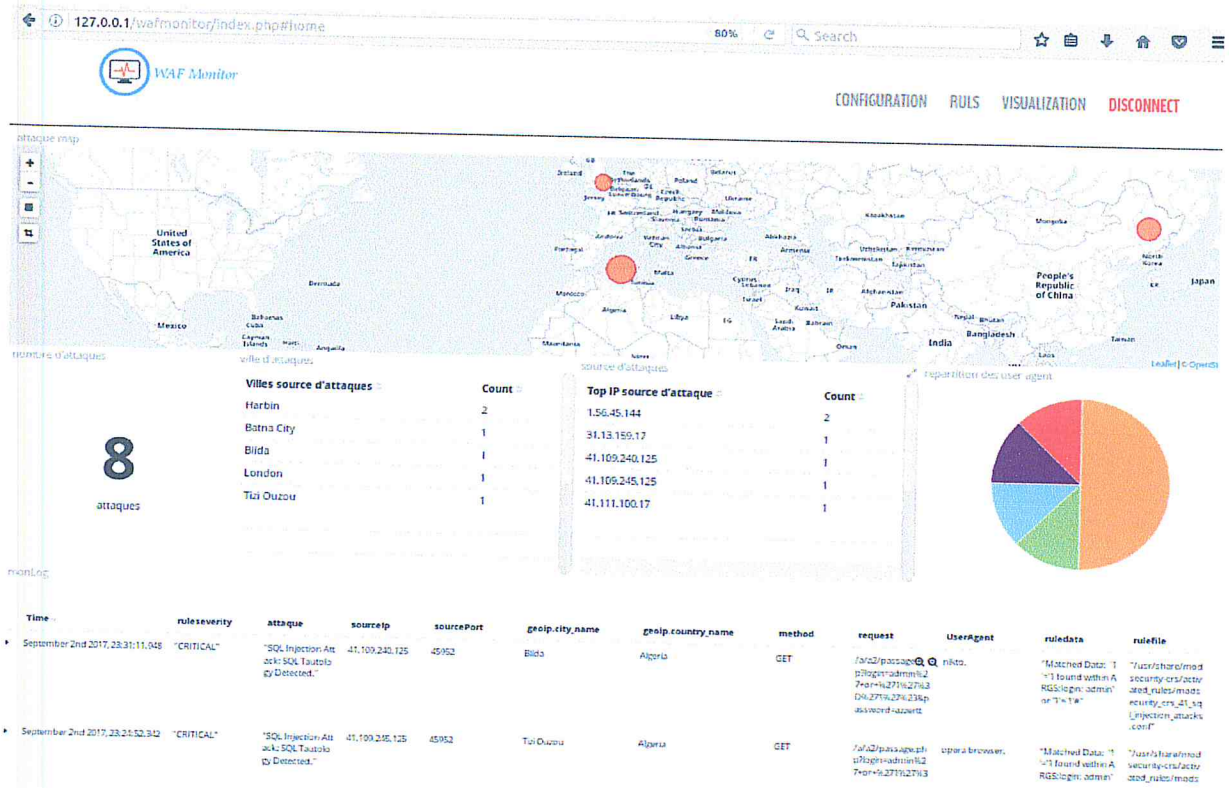


Figure 42 – aperçue global de l'interface de monitoring

Si vous êtes attaqué, il est très important d'avoir une image de ce que votre attaquant essaie de le faire. Est-ce qu'il utilise un script pré-emballé pour essayer d'entrer dans votre serveur ? Est-ce que c'est quelqu'un utilisant des requêtes d'injection SQL fabriquées à la main via un serveur proxy dans un pays étranger ?

Cette partie de l'interface permet de connaitre les détaille d'une attaque bloqué :

Time	ruleseverity	attaque	sourceIp	sourcePort	geoip.city_name	geoip.country_name	method	request	UserAgent
September 2nd 2017, 23:31:11.948	"CRITICAL"	"SQL Injection Attack: SQL Tautology Detected."	41.109.240.125	45952	Blida	Algeria	GET	/a/a2/passage.php?login=admin%27+or+%271%27%3D%271%27%23&password=azzertt	nikto.
September 2nd 2017, 23:24:52.342	"CRITICAL"	"SQL Injection Attack: SQL Tautology Detected."	41.109.245.125	45952	Tizi Ouzou	Algeria	GET	/a/a2/passage.php?login=admin%27+or+%271%27%3D%271%27%23&password=azzertt	opera browser.
September 2nd 2017, 23:16:56.708	"CRITICAL"	"SQL Injection Attack: SQL Tautology Detected."	41.55.19.31	45952	-	South Africa	GET	/a/a2/passage.php?login=admin%27+or+%271%27%3D%271%27%23&password=azzertt	internet explorer.
September 2nd 2017, 23:10:54.319	"CRITICAL"	"SQL Injection Attack: SQL Tautology Detected."	41.55.69.3	45952	-	South Africa	GET	/a/a2/passage.php?login=admin%27+or+%271%27%3D%271%27%23&passwor	chrome google.

Figure 43 – table détaillée sur les attaques

Ici on peut visualiser quelques statistiques comme le nombre d'attaques globales ou le nombre d'attaque par ville :

Figure 44 – exemple d'une visualisation sur les évènements produites

Un aperçue de statistique sur les navigateurs qui sont source d'attaques et les user-agents :

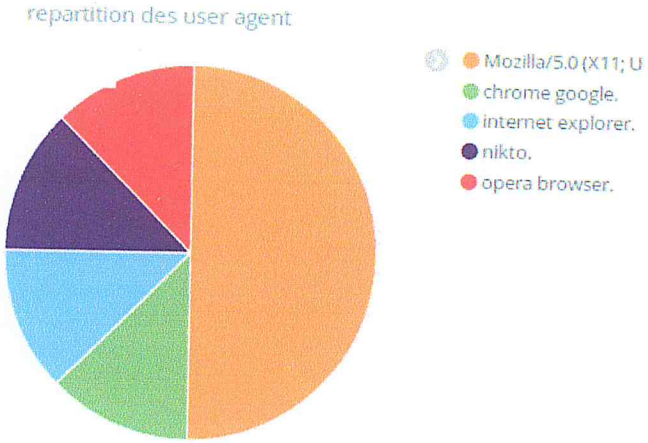


Figure 45 – exemple de statistique sur les user-agents qui sont source d'attaques

Aussi une carte de géolocalisation des adresses IP qui sont source d'attaques :

Figure 46- exemple d'une visualisation sur la source des attaques

4. Conclusion

Dans ce chapitre nous avons à présenter notre environnement de travail dans lequel on a déployé notre système de protection puis nous avons présenté l'implémentation du module de monitoring et d'administration que nous avons ajouté au fonctionnalités du pare-feu web adopté (modsecurité), un aperçue sur les résultats de l'IHM réalisé dans une variétés de visualisations significatifs.

Ici on peut visualiser quelques statistiques comme le nombre d'attaques globales ou le nombre d'attaque par ville :

nombre d'attaques

8

attaques

ville d'attaques

geoip.city_name.keyword: Descending	Count
Harbin	2
Batna City	1
Blida	1
London	1
Tizi Ouzou	1

4. Conclusion

Dans ce chapitre nous avons à présenter notre environnement de travail dans lequel on a déployé notre système de protection puis nous avons présenté l'implémentation du module de monitoring et d'administration que nous avons ajouté au fonctionnalités du pare-feu web adopté (modsecurité), un aperçue sur les résultats de l'IHM réalisé dans une variétés de visualisations significatifs.

Conclusion générale

Assurer la sécurité d'une application Web est une priorité absolue vue quelle est toujours exposée aux attaques, donc il est primordial d'en développer sa sécurité dès le départ, car si le développeur ne porte pas une attention particulière, on peut être sûr que celle-ci comportera des failles exploitables.

Un certain nombre de failles très connues fonctionnent toujours sur de nombreuses applications, cela ne demande aucune compétence particulière pour lancer une attaque, vu la multitude d'outils automatisés faciles d'accès. Il faut donc bien définir ses contraintes de sécurité sur le cahier des charges et ne pas hésiter à former les équipes de développement à la sécurisation de ces types de failles.

Egalement pour assurer la sécurité complète de l'application, la vérification seule du code ne suffit pas, il faut s'assurer que le serveur qui héberge l'application, la base de données, en somme toute l'infrastructure soit correctement configurée. Pour cela il faut compter sur plusieurs mesures de sécurité, afin de minimiser les risques. Chaque mesure de sécurité doit assurer l'intégrité de la couche dont elle est responsable indépendamment des autres couches.

Dans cette optique on a travaillé sur la mise en place d'un modèle de sécurité durant le cycle de développement et la mise en production d'une application web, et à travers ce mémoire on a essayé d'apporter une approche personnalisable qui comprend une politique de sécurité des applications web. Le pare-feu web semble être une très bonne solution pour protéger ces applications vu que c'est une couche virtuelle qui peut combler des lacunes, et de ce fait nous faire gagner du temps et de l'effort perdu dans la rectification du code. Dans ce contexte l'objectif était de contribuer à étendre les fonctionnalités du WAF avec une interface de monitoring et d'administration pour faire le suivi des évènements et offrir un meilleur contrôle et une meilleure surveillance de cet outil. Aussi l'objective étais de construire une base de données centralisé qui offre une connaissance approfondie sur l'ensemble de trafic a notre destination, ceci dans le but de constater l'évolution des incidents à notre destination, et à la fin améliorer la politique de sécurité constamment (liste noire, liste blanche, statistiques...). L'implémentation réalisée est modulaires et tout élément ou composant peut être changé ou amélioré séparément.

Notre perspective est de construire une base de connaissance autour de plusieurs parties tierces qui s'entraident pour sécuriser l'application web (par exemple un nœud de teste de pénétration, un nœud de protection réseau ou de protection de logiciels malveillants). Regrouper la collecte des évènements survenus des différents matériels chargés de la sécurité du système pour arriver à une surveillance et un contrôle général plus pertinent sur l'ensemble de l'infrastructures matérielles et logicielles.

Ce projet a été pour nous une chance et une formidable opportunité pour découvrir un domaine informatique nouveau et très vaste, ce qui nous a permis d'acquérir de l'expérience en sécurité informatique et d'approfondir nos connaissances dans le domaine du développement web sécurisé.

Bibliographie

[1] R. Roure, «Sécurité : pourquoi le WAF est-il indispensable ?,» 19 10 2015. [En ligne]. Available: http://www.orange-business.com/fr/blogs/securite/webtech/securite-pourquoi-le-waf-est-il-indispensable- [Accès le 04 03 2017].

[2] These de master (No /186/2016) Sécurité des systemes d'information – Université houari boumedienne, alger.

[3] N. Cavigneaux, «Les enjeux de la sécurité des applications Web,» 05 07 2011. [En ligne]. Available:https://www.synbioz.com/blog/les_enjeux_de_la_securite_des_applications_web [Accès le 15 03 2017].

[4] M. rouse, «World Wide Web,» 03 2017. [En ligne]. Available: http://whatis.techtarget.com/definition/World-Wide-Web [Accès le 02 04 2017].

[5] D. Stuttard et . P. Marcus, The web application hacker's handbook - second edition finding and exploiting security flaws, Wiley & Sons, 2011, p. 5.

[6] P. Sharma, « internet And WWW Difference »,04 08 2009 . [En ligne]. Available: https://www.techpluto.com/difference-between-internet-and-world-wide-web/

[7] M. Zalewski, The Tangled Web: A Guide to Securing Modern Web Applications, San Francisco, Californie, États-Unis: No Starch Press, 2011, p. 41.

[8] S. Fesnien , «le protocole http,» , bibiotheque voozook[En ligne]. Available: https://www.coozook.com/static/book-samples/B212934F6A-sample.pdf. [Accès le 02 04 2017].

[9] D. Stuttard et . P. Marcus, The web application hacker's handbook - second edition finding and exploiting security flaws, Wiley & Sons, 2011, p. 40.

[10] D. Stuttard et . P. Marcus, The web application hacker's handbook - second edition finding and exploiting security flaws, Wiley & Sons, 2011, p. 41.

[11] Mozilla Foundation, «Méthodes de requête HTTP,» [En ligne]. Available: https://developer.mozilla.org/fr/docs/HTTP/M%C3%A9thode. [Accès le 15 avril 2017].

[12] M. Zalewski, The Tangled Web: A Guide to Securing Modern Web Applications, San Francisco, Californie, États-Unis: No Starch Press, 2011, p. 54 - 56.

[13] D. Stuttard et . P. Marcus, The web application hacker's handbook - second edition finding and exploiting security flaws, Wiley & Sons, 2011, p. 49.

[14] WASC, «web application risks,» [En ligne]. Available: http://projects.webappsec.org/f/WASCTCv2_0.pdf%20risque%20=%20menace%20*vulnirabilt%C3%A9*impact. [Accès avril 2017].

[15] H. SCHAUER, «hsc.fr,» 10 avril 2008. [En ligne]. Available: http://www.hsc.fr/ressources/presentations/netfocus08-iso27/netfocus08-iso27.pdf. [Accès le 25 04 2017].

[16] J.-F. Audenard, «les-5-mins-du-professeur-audenard-episode-14-menace-vulnerabilite-et-impact,» 11 09 2013. [En ligne]. Available: http://www.orange-business.com/fr/blogs/securite/les-5-minutes-du-professeur-audenard/les-5-mins-du-professeur-audenard-episode-14-menace-vulnerabilite-et-impact. [Accès le 15 04 2017].

[17] j. ulloa, «pourquoi un WAF , blog about web application security,» 2009 01 09. [En ligne]. Available: http://johanne.ulloa.org/pourquoi-un-waf.html. [Accès le 20 04 2017].

[18] Owasp and Wasc threats classification : http://owasp.org http://www.webappsec.org/ https://aresu.dsi.cnrs.fr/IMG/pdf/failles_de_securite_v1-3.pdf [Accès 25 04 2017]

[19] these master « cloud based web application firewall » - USDB Génie des Systemes informatique – 2013

[20] agence nationale de sécurité des systèmes d'information, availble : https://www.ssi.gouv.fr/uploads/IMG/pdf/NP_Securite_Web_NoteTech.pdf

[21] MIKE SHEMA. « Hack Notes, Web Security Portable Reference ». 2e ed. (US, California - 2003) P.140

[22] thèse de master « PROPOSITION D'UN SYSTÈME D'AUDIT CODE SOURCE » USDB – Sécurity des systemees informatique – 2017 - p25.

[23] J. Totzek-Hallhuber, «waf-vs-ng-fw-ips-julian-totzek-hallhuber,» 2015 mai 4. [En ligne]. Available: https://www.linkedin.com/pulse/waf-vs-ng-fw-ips-julian-totzek-hallhuber. [Accès le 30 avril 2017].

[24] i. Sans, «the diffrence between an ips ans a web application firewall,» [En ligne]. Available: https://www.sans.org/security-resources/idfaq/what-is-the-difference-between-an-ips-and-a-web-application-firewall/1/25. [Accès mai 2017].

[25] C. Kumar, «5 Open Source Web Application Firewall for Better Security,» 4 septembre 2016. [En ligne]. Available: https://geekflare.com/open-source-web-application-firewall/. [Accès le 1 mai 2017].

[26] S. Wilkins, «Web Application Firewall Guide,» 26 11 2015. [En ligne]. Available: http://www.tomsitpro.com/articles/web-application-firewall-guide,2-988-2.html. [Accès le 18 5 2017].

[27] Jim beechey sans technology institute, «web application fire walls,» 2009. [En ligne]. Available: http://www.sans.edu/student-files/projeccts/200904_01.doc. [Accès le 3 mai 2017].

[28] trustwave : https://www.trustwave.com/Company/SpiderLabs/

[29] nginx : https://nginx.org/

[30] aqtronix webnight :https://www.aqtronix.com/

[31] J. Totzek-Hallhuber, «waf-vs-ng-fw-ips-julian-totzek-hallhuber,» 2015 mai 4. [En ligne]. Available: https://www.linkedin.com/pulse/waf-vs-ng-fw-ips-julian-totzek-hallhuber. [Accès le 30 avril 2017].

[32] web application firewalls evaluation criteria :http://projects.webappsec.org/f/wasc-wafec-v1.0.pdf

[33] modsecuity :http://www.modsecurity.org

[34] T. dostes, «modsecurty 2.x,» 2009. [En ligne]. Available: http://cesar.resinfo.org/cours-prive/ADF-012009/ADF2009_modsecurity.pdf. [Accès le 15 mai 2017].

[35] open web application security projects, «Application Security Verification Standard 3.0.1,» juillet 2016. [En ligne]. Available: https://www.owasp.org/images/3/33/OWASP_Application_Security_Verification_Standard_3.0.1.pdf. [Accès le 2 aout 2017].

[36] Norme ISO IEC 27002-2013 - Code de bonne pratique pour le management de la sécurité de l'information. & ISO IEC 27005-2013 Gestion des risques liés à la sécurité de l'information. [Accès le 22 juin 2017].

[37] Common weakness enumeration: https://cwe.mitre.org/data/definitions/index.html. [Accès le 30 juin 2017].

[38] agence national de sécurité des systèmes d'information (France) - politique de sécurité des applications web . Available : https://www.ssi.gouv.fr/uploads/IMG/pdf/NP_Securite_Web_NoteTech.pdf [Accès le 2 aout 2017].

[39] Direction générale de sécurité des systèmes d'information (Maroc). – politique de sécurité des applications web . [Accès le 4 aout 2017].

[40] SANS institute, Frank Kim, «Securing Web Application Technologies [SWAT] Checklist,» 2013. [En ligne]. Available: https://securingthehuman.sans.org/media/resources/planning/STH-poster-winter-2013.pdf https://software-security.sans.org/resources/swat. [Accès le 12 juin 2017].

[41] B. elior, «Haute Disponibilité et Tolérance de Panne,» 23 7 2015. [En ligne]. Available: https://fr.slideshare.net/EliorBoukhobza/haute-disponibilit-et-tolrance-de-panne-50843503. [Accès le 25 avril 2017].

[42] «Guide de Bonnes Pratiques pour les Administrateurs Systèmes et Réseaux,» [En ligne]. Available: http://gbp.resinfo.org/?p=70. [Accès le 27 avril 2017].

[43] A. Polyakov, «SAP Security Notes September 2015,» 9 septembre 2015. [En ligne]. Available: https://blogs.sap.com/2015/09/09/sap-security-notes-september-2015/. [Accès le 29 4 2017].

[44] F5 written by Alan Murphy and KJ (Ken) Salchow, Jr, «Applied App Security White Paper,» octobre 2007. [En ligne]. Available: https://fr.scribd.com/document/162296679/Applied-App-Security-White-Paper. [Accès le 20 mai 2017].

[45] S. GIORIA, «web application fire walls,» 6 juin 2009. [En ligne]. Available: https://www.cert-ist.com/pub/files/Document_Cert-IST_000333.pdf. [Accès le 2 5 2017].

ANNEXE

OWASP (Open Web Application Security Project):

OWASP est une communauté en ligne évolue dans la sécurité des applications Web. Elle a pour vocation de publier des recommandations de sécurisation Web et de proposer aux internautes, administrateurs et entreprises des méthodes et outils de référence permettant de contrôler le niveau de sécurisation de ses applications Web.

La fondation OWASP est une organisation caritative enregistrée 501(c)(3) aux États-Unis depuis 2004 et enregistrée en Europe depuis juin 2011 en tant qu'Organisation à but non lucratif qui supporte les infrastructures et projets OWASP. OWASP est aujourd'hui reconnue dans le monde de la sécurité des systèmes d'information pour ses travaux et recommandations liées aux applications Web.

Site officiel : owasp.org

SANS Institute (SysAdmin, Audit, Network, Security) :

Sans est une organisation regroupant 165 000 professionnels de la sécurité (consultants, administrateurs système, universitaires, agents gouvernementaux…) ayant pour but de mutualiser l'information concernant la sécurité des systèmes d'information.

Le Sans Institute est également une université de formation aux technologies de sécurité. Elle apparait comme une référence dans la presse spécialisée.

Site officiel : sans.org

WASC (web application security consortium):

Le Web Application Security Consortium (WASC) est un groupe international d'experts, de praticiens de l'industrie et de représentants organisationnels qui produisent des normes de sécurité open source et largement approuvées pour le World Wide Web. En tant que communauté active, Web Application Security Consortium (WASC) facilite l'échange d'idées et organise des projets industriels. WASC diffuse régulièrement des informations techniques, des articles contribués, des directives de sécurité et d'autres documents utiles. Les entreprises, les établissements d'enseignement, les gouvernements, les développeurs d'applications, les professionnels de la sécurité et les fournisseurs de logiciels partout dans le monde utilisent leurs publications pour faire face aux défis présentés par la sécurité des applications Web.

Site officielle : webappsec.org

CLUSIF :

Le CLUSIF, Club de la sécurité de l'information français, est un club professionnel constitué en association indépendante. Ouvert à toutes les entreprises et collectivités, ce club rassemble des Offreurs et des Utilisateurs issus de tous les secteurs de l'économie. L'objectif principal du CLUSIF est de favoriser les échanges d'idées et de retours d'expériences au travers de groupes de travail, de publications et de conférences thématiques. Les sujets abordés, en relation avec la sécurité de l'information, varient en fonction de l'actualité et des besoins des membres de l'association.

Site officielle : clusif.fr

Le principe du fonctionnement des règles selon ModSecurity

Nous pouvons créer des règles et définir à quelle phase elles interviendront, La construction des règles est écrites dans un langage propre à ModSecurity, Le langage des règles est implémenté en utilisant 9 directives, qui sont listées dans le tableau suivant :

Directives	Discription
SecAction	Effectue une action inconditionnelle. Cette directive est essentiellement une règle qui est toujours effectué.
SecDefaultAction	Spécifie la liste d'action par défaut, qui sera utilisée dans les règles qui suivent.
SecMarker	Crée un marqueur qui peut être utilisé conjointement avec l'action skipAfter. Elle crée une règle qui ne fait rien, mais qui lui a été assigné un ID.
SecRule	Crée une règle.
SecRuleInheritance	Contrôle si les règles sont héritées dans un contexte de configuration enfant.
SecRuleRemoveById	Supprime la règle avec l'ID donné.
SecRuleRemoveByMsg	Supprime la règle dont le message correspond à l'expression régulière donnée.
SecRuleScript	Crée une règle implémentée à l'aide language Lua.
SecRuleUpdateActionById	Remplace la liste d'action de la règle d'un ID donné par la liste l'action fournie.

SecRule est la principale directive utilisée, qui est utilisé pour créer des règles et dont la plupart du travail est fait avec.

L'anatomie d'une règle est la suivante :

```
SecRule VARIABLES OPERATOR [ACTIONS]
```

VARIABLES Spécifie les variables qui doivent être testées. Ces variables identifient les parties d'une transaction HTTP avec laquelle chaque règle fonctionne. ModSecurity va extraire des

informations provenant de chaque transaction et la rendre disponible, par des variables, pour être utilisé a traves les règles.

OPERATOR précise l'opérateur utilisé en le faisant précéder du symbole @(ex : lt, gt, ..etc.).

Ils indiquent comment une variable doit être analysée. Un seul opérateur est autorisé par règle.

ACTIONS indique ce que l'on veut faire. Cet argument est optionnel.

Ex : mettre une machine cliente en liste blanche.

```
SecRule REMOTE_ADDR "^10\.126\.100\.85$" phase:1,log,allow
```

Quelques exemples de VARIABLES:

Vriables	Descriptions
ARGS	Traite l'ensemble des arguments, ou l'argument spécifié en paramètre.
AUTH_TYPE	Renvoie le type d'authentification utilisé.
HTTP_REFERER	Indique le site de provenance d'une requête.
REMOTE_ADDR	L'adresse IP de la machine cliente.
REMOTE_HOST	Retourne le nom DNS de la machine cliente si l'option HostnameLookUp est activée.
REQUEST_HEADERS	Permet de travailler sur l'entête de la requête .
REQUEST_BODY	Accède aux contenues du corps de la requête.
REQUEST_FILENAME	Retourne le nom du fichier appelé, sans les éventuels paramètres.
REQUEST_URI	Retourne le chemin d'accès au fichier appelé et les éventuels paramètres.

Les Actions :

Les actions permettent à ModSecurity réagir aux événements. Chaque action appartient à l'un des 5 groupes suivants :

Actions perturbatrices : ModSecurity interceptera les données (ex : allow).

Actions non perturbatrices (ex : auditlog).

Actions de traitement du flux (ex : chain).

Actions périphériques (ex : id, msg, severity).

Actions sur les données : elles sont utilisées pour acheminer les informations traitées par d'autres types d'actions.

Actions perturbatrices précisent ce qu'une règle veut faire lors d'une correspondance. Chaque règle doit être associée à exactement une action perturbatrice. L'action `pass` est la seule exception, car elle permettra au traitement de continuer quand une correspondance se produit. Toutes les autres actions de cette catégorie seront bloquées de manière spécifique.

Actions	Descriptions
`allow`	`Arrêter le traitement d'une ou plusieurs phases restantes.`
`block`	`Indique qu'une règle veut bloquer.`
`deny`	`Bloquer la transaction avec une page d'erreur.`
`drop`	`Fermer la connexion réseau.`
`pass`	`Ne bloquez pas, passez à la règle suivante.`
`proxy`	`Requête proxy vers un serveur web backend`
`redirect`	`redirection de requête vers un autre serveur Web`

Actions de traitement du flux

Les actions de flux montre la façon dont les règles sont traitées au cours d'une phase

Actions	Descriptions
`chain`	`Connectez deux règles ou plus dans une seule règle logique.`
`skip`	`Ignorer une ou plusieurs règles qui suivent.`
`skipAfter`	`Passez à la règle ou au marqueur avec l'ID fournie.`

Actions périphériques fournissent des informations supplémentaires sur les règles. L'information est destinée à accompagner les messages d'erreur afin de mieux comprendre pourquoi ils se sont produits.

Actions	Descriptions
id	Affecter une ID unique à une règle.
phase	Phase d'exécution d'une règle.
msg	Message affiché.
rev	Numéro de révision.
severity	Gravité.
tag	Marque

Actions non perturbatrices

Actions	descriptions
auditlog	journaliser la transaction en cours pour auditer le log.
log	Journaliser Message d'erreur; Implique un auditlog.
logdata	Journaliser les données comme part du message d'erreur.
noauditlog	Ne pas enregistrer la transaction en cours pour auditer le log.
nologtag	Ne pas journaliser un message d'erreur; Implique noauditlog.
sanitiseArg	Supprimer le paramètre d'un requêtes du auditlog.

Actions sur les données : ou les actions variables traitent des variables. Ils vous permettent de définir, modifier et supprimer des variables.

Actions	descriptions
capture	Capturez les résultats en une ou plusieurs variables.
Deprecatevar	Décrémenter une variable numérique dans le temps.
Expirevar	Supprimer une variable après une période de temps.
Initcol	Créer une nouvelle collection persistante.
Setvar	Définir, supprimer, incrémenter ou décrémenter une variable.
Setuid	Associer la transaction en cours à une ID utilisateur d'application (nom d'utilisateur).
setsid	Associer la transaction en cours avec un ID de session d'application.

Quelques exemples d'actions :

Auditlog : journalise la transaction dans le fichier d'audit.

```
SecRule REMOTE_ADDR "^192\.168\.1\.100$" auditlog,phase:1,allow
```

Deny : interrompt le traitement et intercepte la transaction.

```
SecRule REQUEST_HEADERS: User-Agent "nikto" "log, deny, msg:' Scanners Identified' "
```

Dans cet exemple il intercepte la transaction car il détecte le scanner de vulnirabilité Nikto

Drop : coupe immédiatement la connexion TCP.

```
SecRule IP:AUTH_ATTEMPT "@gt 25" log, drop, phase:1, msg: 'Possible Brute Force Attack"
```

Ici l'utilisation de la commande drop pour se protéger d'attaque en force brute. Nous enregistrons les tentatives d'authentifications auprès de notre serveur. Si le client distant essaye de se connecter plus de 25 en fois en moins de 2 minutes, ModSecurity rejettera toute nouvelle tentative de connexion.

Proxy : redirige la requête vers un autre serveur grâce au module proxy d'Apache.

```
SecRule REQUEST_HEADERS:User-Agent "Test" log,proxy:http://www.nowhere.com/
```

Severity : définit la gravité d'une alerte.

```
SecRule REQUEST_METHOD "^PUT$" "id: 340002, rev: 1, severity: 2, msg: 'Restricted HTTP function' "
```

Les niveaux de gravité possibles pour severity sont :

0 = EMERGENCY
1 = ALERT
2 = CRITICAL
3 = ERROR
4 = WARNING
5 = NOTICE
6 = INFO
7 = DEBUG

Status : spécifie le status HTTP retourné au client.

```
SecDefaultAction log, deny, status:403, phase:1
```

Le code erreur HTTP 403 correspond à une action interdite. Le serveur HTTP comprend la requête, mais refuse de la traiter. Ce code est généralement utilisé lorsqu'un serveur ne souhaite pas indiquer pourquoi la requête a été rejetée.

Les Operateurs

Cette section indique quelques exemples des opérateurs utilisés dans ModSecurity.

Opérateurs sur les chaînes de caractères

Operateur	descriptions
`@beginsWith`	Commence par.
`@contains`	Contient.
`@endsWith`	Se termine par.
`@rx`	Expression régulière égale à.
`@streq`	Chaîne égale à.
`@within`	Dans.

Opérateurs numérique

Operateur	descriptions
`@eq`	Égal
`@ge`	Plus grand ou égal
`@gt`	Plus grand que
`@le`	Plus petit ou égal
`@lt`	plus petit que

Divers operateurs

Operateur	descriptions
`@geoLookup`	Détermine l'emplacement physique d'une adresse IP
`@inspectFile`	Invoque un script externe pour inspecter un fichier
`@verifyCC`	Vérifie si le paramètre est un numéro de carte de crédit

Exemples :

BeginsWith : Renvoie true si le un certain paramètre en chaîne de caractère se trouve au début d'une entrée.

Exemple : # Il détecte la ligne de requete qui ne commence pas avec " GET"

```
SecRule REQUEST_LINE "!@beginsWith GET" "id:149
```

eq : Effectue une comparaison numérique et renvoie true si la valeur d'entrée est égale au paramètre fourni.

Exemple : # Détecte exactement les requête avec 15 entête

```
SecRule &REQUEST_HEADERS_NAMES "@eq 15" "id:153"
```

La configuration de ModSecurity

Le fichier **modsecurity_crs_10_config.conf** définit le comportement par défaut de l'outil :

SecRuleEngine :

On : interception des échanges et application des règles de filtrage.

DetectionOnly : application des règles sans interception.

Off : désactivation de la surveillance.

SecRequestBodyAccess On : activation des règles permettant l'inspection du corps des requêtes.

SecResponseBodyAccess On : même chose pour le corps des réponses.

SetDefaultAction : permet de déterminer l'action par défaut si aucune règle ne s'applique.

Nous pouvons spécifier plusieurs actions par défaut :

```
SecDefaultAction log, auditlog, deny, status:403, phase: 2, t:lowercase
```

auditlog : toute transaction est conservée dans le fichier d'audit de ModSecurity.

Status:403 : précise le code HTTP retourné au client dès qu'une règle correspond.

T:lowercase : active la transformation de tous les caractères d'une requête en minuscules.

Il est possible de préciser les informations d'audit conservées :

SecAuditEngine :

On : enregistre toutes les transactions par défaut.

Off : n'enregistre aucune transaction.

RelevantOnly : conserve seulement les transactions à l'origine d'une alerte ou d'une erreur.

SecAuditLog : définit l'emplacement du fichier de traces.

SecAuditLogPart : spécifie quelles parties des échanges sont enregistrées :

A : informations spécifiques aux journaux d'événements.

B : les entêtes de la requête.

I : le corps de la requête (à l'exception des fichiers).

F : les entêtes de la réponse.

Z : les informations signifiant la fin de l'entrée dans les journaux.

ModSecurity permet d'enregistrer beaucoup plus d'informations que le serveur Apache dans les fichiers de traces. Nous disposons ainsi d'informations d'audit beaucoup plus détaillées. Voyons comment définir ce que nous souhaitons conserver.

D'autres directives :

• Définition de limites sur les volumes de données traités :

- **SecResponseBodyLimit**
- **SecRequestBodyInMemoryLimit**

• Définition du niveau de logs : **SecDebugLogLevel**

0 : aucun enregistrement.

1 : enregistrement des erreurs.

2 : alertes.

3 : notifications (valeur par défaut sous Debian).

4 : détails dans le traitement de la transaction.

5 : équivalent au niveau 4 en plus détaillé.

9 : tout est enregistré.

Un outil de prise d'empreinte n'aura aucun mal à retrouver la véritable signature du serveur.
SecResponseBodyLimit : taille maximale autorisée (en bytes) pour le corps d'une réponse.
SecResponseBodyInMemoryLimit : taille maximale autorisée pour stocker en mémoire le corps d'une requête.

Contenu Règles de base

Les règles de base de Modsecurity sont organisées en fonction du type d'attaque ou de protection, Elles sont stockées dans plusieurs fichiers et L'ordre d'application des règles importe :

```
modsecurity_crs_10_config.conf
modsecurity_crs_20_protocol_violations.conf
modsecurity_crs_21_protocol_anomalies.conf
modsecurity_crs_23_request_limits.conf
modsecurity_crs_30_http_policy.conf
modsecurity_crs_35_bad_robots.conf
modsecurity_crs_40_generic_attacks.conf
modsecurity_crs_45_trojans.conf
modsecurity_crs_50_outbound.conf
```

Passons en revue la fonction de chacun de ces fichiers :

mod_security_crs_10_config :

- fichier de configuration du moteur,
- définition des comportements généraux.

mod_security_crs_20_protocol_violations:

- règles vérifiant que les requêtes sont conformes au protocole HTTP.
- intervention en phase 2 du cycle de vie de ModSecurity.
- élimination d'un grand nombre d'attaques automatisées.

mod_security_crs_21_protocol_anomalies:

- recherche de motifs synonymes d'attaques HTTP.
- intervention en phase 2.
- rejet des transactions concernées.

mod_security_crs_23_request_limits:

- limitation du nombre et de la taille des arguments soumis lors d'une requête HTTP.

- intervention en phase 2.
- protection contre les attaques de type buffer overflow ou manipulation des paramètres.

mod_security_crs_30_http_policy :

Règles restreignant l'utilisation du protocole HTTP par les clients, intervention en phase 2.

- blocage de méthodes (CONNECT, DEBUG, TRACE).
- refus du protocole HTTP 0.9.
- interdiction de fichiers dont l'extension est dangereuse (.ini, .key, .old).

mod_security_crs_35_bad_robots:

- limitation des nuisances générées par des robots d'indexation qui n'en seraient pas
- enregistrements des transactions, mais pas de blocage
- intervention en phase 2.
- détection des scanners de sécurité.

mod_security_crs_40_generic_attacks:

- protection des sessions.
- injections de code SQL.
- attaques de type Cross Site Scripting (XSS).
- injections diverses (PHP, LDAP, etc.).

mod_security_crs_45_trojans :

- détection des tentatives d'accès aux trojans et portes dérobées déjà installées
- intervention en phase 2.
- la mise à jour régulière des ces règles est indispensable.
- attention : contournement possible de ces filtres.

mod_security_crs_50_outbound :

- détection et interception de codes erreurs trop explicites renvoyées par une application
- intervention en phase 4.
- renvoie le code erreur HTTP 501 (opération non supportée par le serveur).

Les règles de base peuvent bloquer le fonctionnement de certaines applications, ModSecurity nécessite donc un important travail de mise au point pour personnaliser les règles.