



**HAL**  
open science

## Filtrage Efficace Pour la Détection d’Intrusions

Tarek Abbes, Adel Bouhoula, Michaël Rusinowitch

► **To cite this version:**

Tarek Abbes, Adel Bouhoula, Michaël Rusinowitch. Filtrage Efficace Pour la Détection d’Intrusions. Conférence Francophone sur Sécurité et Architecture Réseaux (SAR’03), Jul 2003, Nancy, France, 10 p. inria-00099496

**HAL Id: inria-00099496**

**<https://inria.hal.science/inria-00099496v1>**

Submitted on 26 Sep 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Filtrage Efficace Pour la Détection d’Intrusions

Tarek Abbes et Adel Bouhoula et Michaël Rusinowitch

LORIA, 615, Rue du Jardin Botanique, BP 101, 54602 Villers les Nancy Cedex  
SUP’COM, Cité Technologique des Communications 2083 Ariana Tunisie

---

Les systèmes de détection d’intrusions (IDS) sont indispensables pour compléter la protection des pare-feux. Cependant, ils ont des difficultés à traiter le haut débit et à mener parallèlement une analyse précise sur le contenu des paquets. Nous proposons dans cet article une nouvelle approche pour analyser le trafic réseau. Nous nous appuyons sur une organisation intelligente des règles de détection et sur des algorithmes de recherche simultanée de signatures. Cette méthodologie a été implantée dans le système de détection d’intrusions “Snort”.

**Mots clés:** Détection d’Intrusions, Evasion, Snort, Filtrage

---

## 1 Introduction

Un système de détection d’intrusions analyse l’activité interne et externe du réseau et identifie les scénarios d’attaques susceptibles de compromettre l’intégrité du système informatique. Son déploiement est indispensable afin de protéger convenablement le réseau et combler les lacunes des pare-feux. Il existe deux grandes familles de méthodes de détection d’intrusions. La première famille procède par analyse comportementale (anomaly detection). Elle consiste à modéliser les comportements normaux du système et ses utilisateurs. Une déviation par rapport au comportement normal peut révéler une attaque. Le nombre de fausses alertes engendrées est important mais la méthode peut révéler de nouvelles attaques non connues auparavant. Les techniques généralement employées pour modéliser le comportement se fondent sur des outils statistiques et des algorithmes d’apprentissage automatique.

La deuxième famille s’appelle “analyse par scénario” ou “misuse detection” et elle stocke dans des bases de données les signatures des attaques connues. Le nombre de faux positifs est diminué considérablement par rapport à la première méthode mais cette technique se restreint aux attaques déjà connues. Les systèmes de détection d’intrusions (IDS) analysent les fichiers d’audit pour retrouver les instances des scénarios connus. Les techniques d’analyse actuelles utilisent le filtrage (pattern matching), les systèmes experts, les réseaux de Petri, les algorithmes de “model-checking”, etc.

Les IDS forcent les attaquants à compliquer leurs plans d’attaques afin de camoufler leurs activités malveillantes. Actuellement, ceux-ci envoient les paquets en désordre et exploitent le comportement des systèmes d’exploitation à gérer les paquets superposés afin de construire des suites de paquets dont la reconstitution conduit à une attaque. Ils emploient par ailleurs des techniques d’insertion qui consistent à envoyer des paquets qui seront ignorés par l’hôte. Une autre technique d’évasion consiste à envoyer une attaque sur plusieurs paquets mais dont certains seront ignorés par l’IDS.

Nous nous sommes intéressés dans cet article à la détection par scénario et plus particulièrement à l’application des techniques de reconnaissance de motifs pour retrouver les signatures d’attaques. D’abord, nous avons regroupé et ordonné les règles d’IDS pour pouvoir appliquer efficacement les méthodes de reconnaissance simultanée de plusieurs motifs. Ensuite, nous avons implémenté et testé trois algorithmes de filtrage multiple. Enfin, nous avons proposé une nouvelle technique de détection, efficace pour contrarier les techniques d’évasion. Afin de valider nos idées, nous avons implanté nos algorithmes dans le système de détection d’intrusions Snort développé initialement par Martin Roesch[Roe99].

Dans la Section 2 de cet article, nous introduisons le système Snort et ses techniques de détection d’intrusions. Ensuite, nous expliquons la méthode utilisée pour regrouper les règles de Snort. Nous

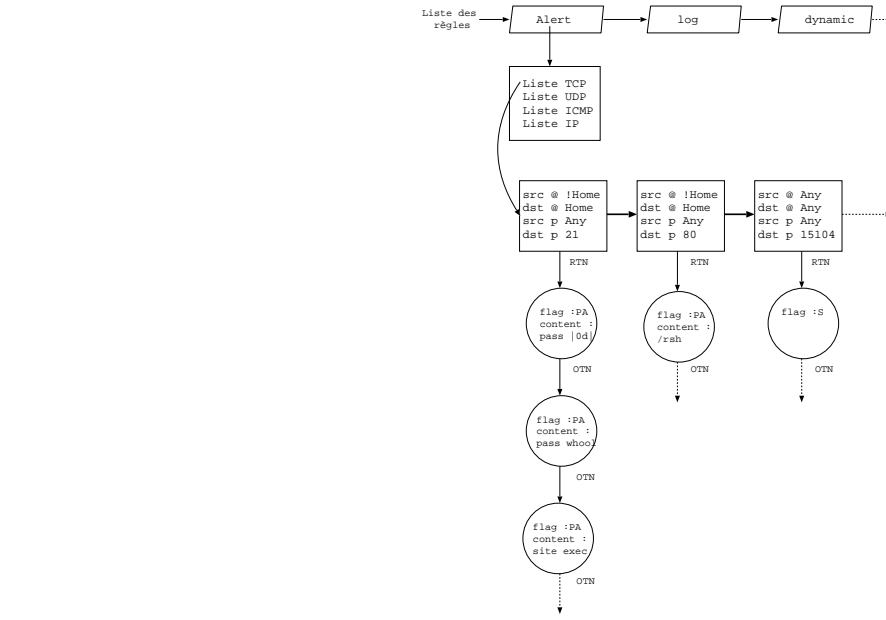


Fig. 1: Chaîne de détection de Snort

présentons dans la Section 4 les algorithmes de reconnaissance de plusieurs motifs que nous avons implanté dans Snort. La Section 5 met en œuvre notre nouvelle procédure pour analyser le trafic réseau. Enfin, nous concluons notre travail dans la Section 6.

## 2 Introduction à Snort

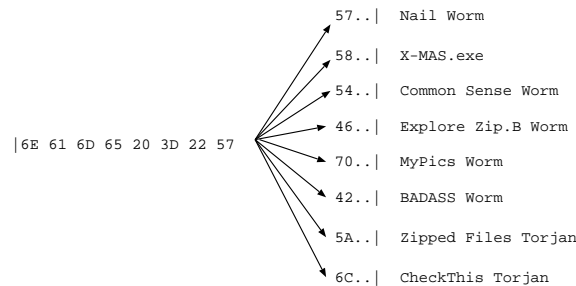
Snort appartient à la famille des détecteurs d'intrusions par scénarios. Il utilise une base de règles spécifiée dans un simple fichier de configuration. Les règles peuvent être classées par type de trafic. Ainsi, on trouve des règles spécifiques aux serveurs de noms (DNS), aux appels de procédures distants (RPC), aux services web, etc. L'administrateur configure le système de détection d'intrusions pour satisfaire la politique de sécurité du réseau qu'il administre. Il sélectionne alors les règles qui décrivent les caractéristiques des mauvais trafics contre lesquels il est indispensable de se protéger. Il peut par ailleurs formuler ses propres règles pour analyser un type de trafic particulier. Un exemple de règles se présente ainsi :

```
alert tcp !$HOME any → !$HOME 23 (msg: "Telnet Livingston DoS",
  flags :PA ;content : {fff3 fff3 fff3 fff3 fff3}) (1)
```

Snort commence la phase de pré-traitement par une analyse syntaxique des règles sélectionnées. Il construit une liste chaînée à 3 dimensions (Fig. 1), prenant en compte le type de trafic (TCP, ICMP, UDP, IP) et les caractéristiques de la connexion (adresse source, adresse destination, port source, port destination). Chaque règle sera représentée par une OTN (Option Tree Node) qui sauvegarde les options de cette règle (partie italique de la règle 1). Ces options servent à la détection (valeur du TTL : time to live, valeur du TOS : type of service, valeur ID : identifiant du paquet IP,...), au déclenchement des réponses suite à la détection des attaques (contenus des mots clés REACT et RESP) et aux opérations d'écriture dans les fichiers logs (LOGTO : nom du fichier log, msg : description de l'attaque,...). L'OTN est attachée à une RTN (Rule Tree Node) partagée par plusieurs OTN et qui exprime la partie tête de la règle (partie non italique de la règle 1).

Après avoir construit la liste des règles et initialisé les interfaces réseaux, les plugins de détection, du post traitement, et d'affichage, Snort commence son écoute sur les interfaces réseaux spécifiées. Il utilise pour cela la bibliothèque pcap et envoie chaque paquet intercepté à la chaîne de détection. Il sélectionne ensuite les règles adéquates et utilise leurs motifs pour reconnaître les signatures d'attaques.

Snort utilisait dans ses premières versions la méthode naïve pour retrouver les motifs. Cette technique de reconnaissance a été améliorée depuis. D'abord, l'algorithme de Boyer-Moore a permis d'accélérer énormément la recherche. Vu les grandes similarités observées dans les motifs d'attaques (Fig. 2), l'intérêt du filtrage multiple est apparu clairement. Varghese et Fisk [MF01] ont souligné l'importance de l'opération de filtrage dans Snort qui occupe désormais 31% du total du temps d'exécution sur leurs corpus de tests. Afin d'optimiser cette opération, ils ont introduit la méthode de filtrage de Boyer-Moore Horspool (BMH) et deux autres méthodes de filtrage multiple : la machine d'Aho-Corasick et l'algorithme Set-Wise Boyer-Moore. Ils ont réussi à réduire le temps moyen de recherche d'un facteur de 4.6. Les deux algorithmes de filtrage multiple donnent presque les mêmes résultats. Les auteurs suggèrent l'utilisation dynamique des 3 méthodes de reconnaissance. En effet, le BMH présente les meilleurs résultats lorsqu'il s'agit d'un seul motif. En revanche, si l'ensemble de motifs est inférieur à 100 alors l'utilisation du Set-Wise Boyer-Moore est la plus optimale. Tandis qu'avec un ensemble de motifs supérieur à 100, l'algorithme d'Aho-Corasick semble être le plus adéquat.



**Fig. 2:** Similarités dans les motifs d'attaques

Varghese et Fisk [MF01] ont rassemblé les motifs de toutes les règles appartenant à la même RTN. La disposition actuelle des OTN dans Snort les empêche de mieux factoriser les règles. Cette opération est vitale afin d'optimiser la recherche dans la liste des règles. Elle permet aussi la construction efficace d'un ensemble de motifs pour un groupe bien déterminé de règles. Pour assurer cette factorisation, nous avons introduit une nouvelle méthode de compilation des règles de Snort. Cette méthode, permet de rassembler un groupe de règles ayant des caractéristiques communes.

### 3 Restructuration des règles de Snort

Nous envisageons dans cette section de tenir compte des parties communes des règles de Snort afin d'optimiser la recherche. Le résultat de cette factorisation est la création de deux nouveaux types d'OTN, appelés OTN primaire et OTN secondaire. Les OTN primaires rassemblent les parties communes entre diverses règles. Les autres options spécifiques à chaque règle sont représentées maintenant dans une OTN secondaires. Dans cette nouvelle architecture, nous appelons OTN ordinaire, chaque OTN qui représente une règle n'ayant pas subi de factorisation. Cet OTN correspond parfaitement à l'ancienne notion d'OTN.

Supposant que nous disposons des 4 règles suivantes :

```
R1: Alert TCP @src prt_src → @dst prt_dst (msg : 'α', flags:A+, ttl:1, sid:x_1)
R2: Alert TCP @src prt_src → @dst prt_dst (msg : 'β', flags:A+, ttl:2, sid:x_2)
R3: Alert TCP @src prt_src → @dst prt_dst (msg : 'δ', flags:A+, ttl:3, sid:x_3)
R4: Alert TCP @src prt_src → @dst prt_dst (msg : 'θ', seq:0, sid:x_4)
```

Les quatre règles ont le même type "alert" et traitent le même flux "TCP". Elles ont par ailleurs la même adresse source, adresse destination, port source et port destination. Elles appartiennent par conséquent à la même RTN. Le schéma classique de disposition de ces règles avec Snort est dressé dans la Fig.3. En optant pour une factorisation d'états, nous obtenons une nouvelle physionomie de la chaîne d'OTNs. Le nombre de vérification de l'option "flags" est réduit de 3 à 1. Ce gain sera plus important avec l'augmentation du nombre de règles.

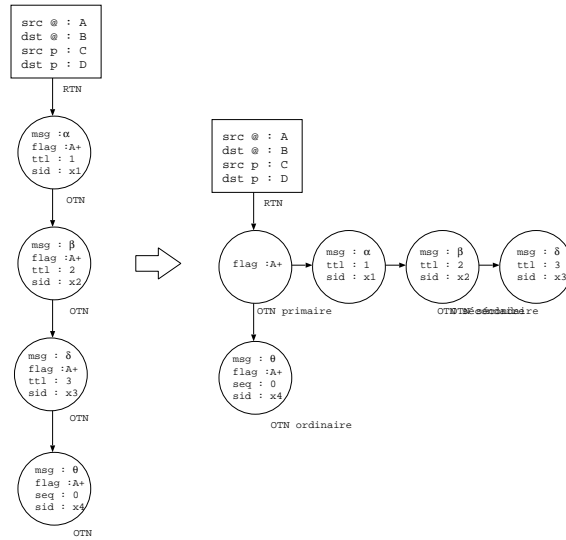


Fig. 3: Factorisation des règles de Snort

D'autre part en rassemblant les champs, nous pouvons introduire un ordre sur les options, ce qui nous mène à affecter des précédences entre les règles. D'autres avantages seront expliqués au fur et à mesure dans ce papier, avantages qui sont en relation avec le filtrage de Snort.

### 3.1 Factorisation des règles de Snort

Soit  $D = \{x_1, x_2, \dots, x_i, \dots, x_n\}$  l'ensemble d'options dans Snort. Nous voulons regrouper les règles selon l'ensemble  $Y = \{y_1, y_2, \dots, y_i, \dots, y_m\} \subseteq D$  des mots clés. Nous pouvons obtenir alors jusqu'à  $m$  classes différentes.

Nous assignons à chaque  $x$  de  $D$  un poids  $p_i$  non nul si  $x_i \in Y$  et 0 sinon. Nous notons  $p = (p_1, p_2, \dots, p_i, \dots, p_n)$  le vecteur résultant.

Chaque règle peut être projetée sur la base  $x = (x_1, x_2, \dots, x_i, \dots, x_n)$  des options de Snort. Nous obtenons alors un vecteur  $v = (e_1, e_2, \dots, e_i, \dots, e_n)_x$  tel que  $e_i = 1$  si l'option  $x_i$  existe dans la règle  $A$  et  $e_i = 0$  dans le cas contraire. Le poids total d'une règle  $r$ , donné par  $F(r)$ , est la somme des poids des options qu'elle contient.

Il est évident que tous les éléments d'une classe présentent le même poids c'est à dire  $F(A) = F([A])$ . En outre, il est judicieux d'assurer l'injectivité de  $F$ . Ainsi, à chaque poids correspond une seule classe possible ce qui nous permet de classer les règles selon leurs poids respectifs. Afin d'assurer l'injectivité de  $F$  il suffit de choisir convenablement  $p$  (par exemple prendre  $p = e^i$ ).

### 3.2 Adaptation de la chaîne de détection

L'IDS peut s'adapter dynamiquement et d'une façon autonome à son environnement d'exécution. En effet, une analyse des alarmes déclenchées met en relief les signatures fréquemment utilisées. Il suffit alors d'extraire périodiquement les options de ces règles et de leur attribuer des poids plus importants. Les règles auront des poids totaux plus grands et se replaceront au début de la chaîne de détection. On optimise ainsi le temps des paquets en fonction de l'activité courante du réseau.

## 4 Reconnaissance multiple dans Snort

L'opération du filtrage est considérée comme la plus pénalisante dans le cycle de détection [Roe99]. L'emploi de la reconnaissance simultanée de plusieurs motifs permet d'optimiser la recherche en profitant des similarités de certains motifs (Fig.2). Par ailleurs, nous pouvons retarder cette opération après la vérification

des options communes des règles ce qui évite parfois le déclenchement inutile du filtrage suite à la violation d'un champ dans la partie commune. Nous avons opté pour le regroupement par défaut de toutes les règles contenant les options "content" et "listcontent". Ces deux champs, ainsi que leurs dérivés (offset, depth, distance, within) sont incorporés alors dans des OTN primaires avec une mise à jour de l'ensemble de motifs.

Nous avons implémenté 3 méthodes de reconnaissance, à savoir Forward DAWG, la machine d'Aho-Corasick et le MATCH-DAWG. La méthode d'Aho-Corasick a été déjà appliquée par Vaghese et Fisk [MF01], mais nous l'avons ré-implantée avec des structures de données semblables à celles utilisées dans le Forward DAWG, pour faciliter la combinaison des deux méthodes et réaliser le MATCH-DAWG. Nous détaillons dans la suite ces trois méthodes en précisant leurs avantages.

#### 4.1 Recherche avec les DAWG

Nous utilisons les graphes acycliques de mots (DAWG) comme une structure de base. Il s'agit en effet d'une structure de données compacte pour représenter les sous séquences des mots (voir par exemple [KR97]).

L'opération de reconnaissance de motifs avec les DAWG consiste à chercher pour chaque caractère du texte, soit  $t[i]$ , et à l'état  $[u]$ , une éventuelle transition étiquetée par  $t[i]$ . Si la transition existe (représentée en trait contenu dans la Fig. 4), alors l'état courant est mis à jour. Sinon on parcourt la chaîne des pointeurs de suffixes (représentées en pointillé dans la Fig. 4) jusqu'à retrouver un état avec une transition  $t[i]$  (ou atteindre l'état initial). La longueur de la partie du mot reconnu sera mise à jour en fonction des transitions effectuées et lorsqu'on atteint une longueur égale à la taille du motif, on s'arrête avec succès. La recherche des motifs avec les DAWG est linéaire en taille du motif et du texte.

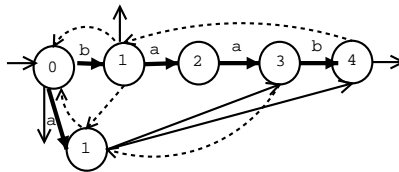


Fig. 4: DAWG du mot "baab"

#### 4.2 Recherche avec la machine d'Aho-Corasick

L'algorithme d'Aho-Corasick peut être vu comme une généralisation de l'algorithme de Knuth-Morris-Pratt pour faire une recherche multiple. Les états de l'automate de recherche sont représentés par des nœuds de l'arbre des préfixes de chaque motif. La construction de cette machine se déroule en deux étapes. D'abord, on construit les états et les transitions directes qui mènent à la reconnaissance du motif. Ensuite, on calcule une fonction d'échec qui correspond au cas de non reconnaissance d'un caractère dans un état donné. La fonction de transition finale sera la combinaison des transitions directes et des transitions d'échec. La recherche du motif s'effectue ensuite en lisant le texte sur le graphe construit. Lorsqu'on atteint un état final, on déclare retrouver le motif correspondant.

#### 4.3 Recherche avec MATCH-DAWG

Cette méthode combine 2 algorithmes à savoir la méthode d'Aho-Corasick et le Reverse Factor. La stratégie consiste à lire de droite à gauche un segment du texte tant qu'il s'agit d'un facteur d'un des motifs. Ensuite, en utilisant le fait que cette partie du texte est contenue dans un motif, on le lit avec la machine à états finis d'Aho-Corasick du gauche à droite afin de reporter les motifs trouvés et calculer la longueur des sauts (Fig.5). Cet algorithme a été introduit par Lecroq dans [LCC93].

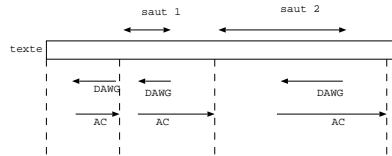


Fig. 5: Exécution de l'algorithme MATCH DAWG

## 5 Détection d'intrusions à la volée

### 5.1 Lutte contre l'évasion

Les attaquants utilisent divers moyens pour contourner les IDS[PN98]. Des préprocesseurs ont été introduits dans Snort afin de contrarier cette nouvelle génération d'attaques. On distingue en particulier les préprocesseurs Frag2 et Stream4 [RG02]. Frag2 se limite par défaut à 4 MB de mémoire et 60 secondes de timeout pour rassembler les fragments IP. D'autre part, Stream4 est configuré par défaut avec une limite de 8 MB de mémoire et un timeout de 30 secondes pour rassembler les flux TCP.

Il est indéniable que ces préprocesseurs ont amélioré le niveau de sécurité assurée par Snort. Cependant, le stockage des paquets dans un espace mémoire et la définition des timeouts peuvent rendre l'IDS vulnérable aux attaques par déni de services et introduisent des retards dans la procédure de détection. Par ailleurs, l'allocation de l'espace mémoire est tributaire des services supportés alors que les timeouts varient selon la capacité du réseau et la quantité de trafic véhiculée. En outre, les attaquants peuvent se synchroniser avec les timeouts pour réussir leurs exploits. Il en résulte que la fixation de ces deux paramètres est une tâche assez cruciale.

Nous avons essayé d'introduire une nouvelle méthode afin de supporter la fragmentation et la segmentation du trafic. L'idée consiste à traiter le trafic dès sa réception mais en sauvegardant le résultat de l'analyse (appelée trace) plutôt que les paquets proprement dits. Cela nous oblige à adapter nos méthodes de reconnaissance de motifs pour supporter cette stratégie. Une bonne gestion des listes de traces est aussi indispensable afin d'éliminer les traces devenues de l'information inutile pour la suite du processus de détection. Effectivement, chaque paquet qui arrive sera analysé pour en déduire 5 propriétés :

- P1 : Le paquet contient une signature d'attaque.
- P2 : Le paquet est un sous mot d'une signature d'attaque.
- P3 : Le paquet contient un suffixe qui est un préfixe d'une signature d'attaque.
- P4 : Le paquet contient un préfixe qui est un suffixe d'une signature d'attaque.
- P5 : Le paquet ne comporte aucune partie d'une signature d'attaque.

On se concentrera dans un premier temps au cas d'une seule signature d'attaque à chercher. On généralisera ensuite la méthode pour le cas de plusieurs signatures.

### 5.2 Recherche d'une seule signature d'attaque

Nous engendrons pour chaque paquet à traiter, une structure appelée trace, qui contiendra les résultats du filtrage. Nous sauvegarderons en particulier :

- Lq : Liste contenant les coordonnées des sous mots du motif.
- Ls : Liste contenant les tailles des préfixes qui sont des suffixes du motif.
- Lp : Liste contenant les tailles des suffixes qui sont des préfixes du motif.

Nous dérivons à partir de ces connaissances des règles de détection. Par exemple, si nous avons un préfixe de taille  $q$ , alors il suffit pour reconnaître tout le motif, d'avoir un suffixe ou une liste de sous mots et un suffixe, successeurs du paquet, ordonnés et de taille totale égale à  $|\text{sig}| - q$  avec  $|\text{sig}|$  la taille de la signature.

D'autre part il est judicieux de sauvegarder les diverses informations contenues dans le paquet et qui sont vitales pour la reconstruction du flux. On s'intéresse en particulier à la taille du paquet ( $l$ ), le numéro de séquence ( $SN$ ), l'identifiant du paquet ( $ID$ ), l'offset de l'entête IP ( $\text{offset}$ ) et le bit du dernier fragment ( $MF$ ).

Dans le cas d'un flux TCP, une autre information s'avère intéressante qui est le premier numéro de séquence. Cette valeur représentée par  $SN_0$  sera fixée dès l'achèvement de l'opération de Three-Way Handshake et sert de repère pour pouvoir ordonner les paquets. Par ailleurs, les numéros des séquences des paquets contenant les drapeaux RST et FIN seront déterminants pour repérer la fin de la liste.

Nous calculons pour chaque paquet qui arrive les différentes listes  $L_q$ ,  $L_s$  et  $L_p$ . Nous gèrerons ensuite les règles de détection et nous insèrerons la trace du paquet dans la liste des traces triée par numéro de séquence contenu dans l'entête TCP ou l'offset du paquet IP (cas de la fragmentation). Si le paquet possède un successeur ou un prédécesseur immédiat alors ces règles seront évaluées. Elles peuvent être éliminées si elles ne sont pas satisfaites comme elles peuvent se transformer et transiter vers d'autres états pour de futures évaluations (Fig.6).

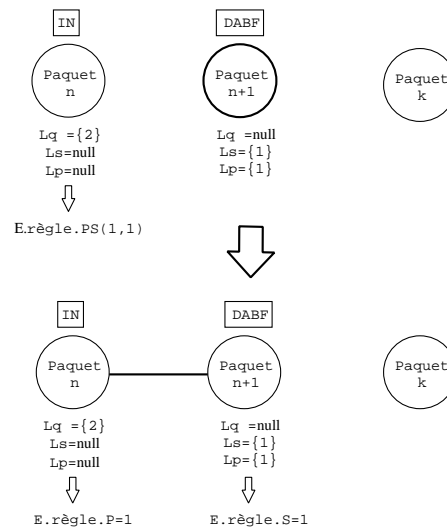


Fig. 6: Transition des règles

Cette méthode de détection analyse uniquement la trace du nouveau paquet reçu après la réception des règles des traces voisines. Nous avons introduit alors la notion de jeton pour exprimer la conjonction entre les règles retenues dans des traces séparées. En effet, le jeton est une variable partagée par plusieurs règles et qui sera libérée partiellement par chaque règle ayant été vérifiée et définitivement si une seule règle détient le jeton. Si une règle n'est pas vérifiée, alors son jeton sera libéré ce qui entraîne l'élimination de toutes les règles qui lui sont attachées.

### 5.3 Gestion des listes de traces

Nous avons créé 3 listes de traces pour les flux TCP, UDP et ICMP. Ces listes sont à deux dimensions et contiennent des nœuds primaires et des nœuds secondaires. Les nœuds primaires sauvegardent les caractéristiques de chaque flux. Ainsi, nous aurons les champs adresse source, adresse destination, port source et port destination pour les flux TCP et UDP et uniquement les adresses source et destination pour le trafic ICMP. Les nœuds secondaires représentent les traces et sont attachés aux nœuds primaires correspondants. L'insertion dans la liste des nœuds secondaires se déroule en respectant l'ordre des traces afin d'assurer le transfert correct des règles associées. Ensuite, si l'ensemble des règles d'une trace  $T$  est vide et les deux traces qui la suivent et la précèdent immédiatement existent, alors nous pouvons éliminer la trace  $T$  et chaîner



directement le prédécesseur au successeur. Le transfert de règles s'établit toujours entre nœuds voisins et de l'ancienne trace vers la nouvelle. Ce sens est naturel puisque la sémantique d'une règle d'écrit ce qu'il suffit de détecter chez les voisins immédiats pour reconnaître le motif d'attaque. Quelques scénarios de gestion de traces sont schématisés dans la figure 7.

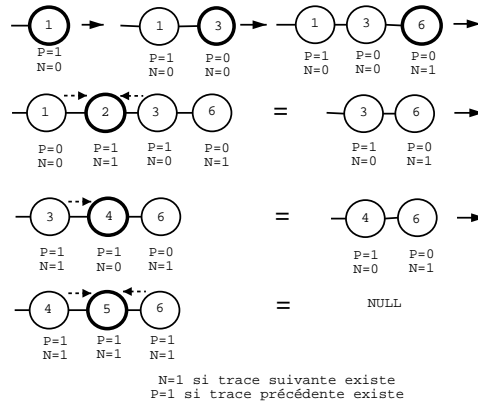


Fig. 7: Gestion des listes de traces

### 5.4 Recherche de plusieurs signatures d'attaques

Notre méthode de recherche d'une signature s'applique aussi à plusieurs signatures à condition d'adapter les algorithmes de filtrage et les structures de données représentant les connaissances et les règles.

Supposons que nous disposons de  $k$  signatures, nous modifions alors nos structures de données de telle sorte que  $E.Tab\_Lq$  représente un tableau de taille  $k$  contenant des listes des coordonnées des sous mots de chaque signature.  $E.Tab\_Lp$ ,  $E.Tab\_Ls$  et  $E.r\grave{e}gle$  sont de même des tableaux de taille  $k$  et contenant respectivement pour chaque signature, les tailles des préfixes, les tailles des suffixes et les champs  $S$ ,  $P$  et  $PS$  des règles.

### 5.5 Expérimentations

Nous avons testé notre méthode de détection sur des trafics simulés et des trafics réels. Le but du trafic simulé était de valider la méthode de reconnaissance de motifs en essayant de détecter des phrases connues insérées dans des segments TCP et des paquets IP envoyés en désordre. Le trafic simulé comporte également un grand nombre de fragments et de longues sessions TCP afin de tester l'influence de ce type de trafic sur nos algorithmes. Nous résumons dans le Tab. 1 les caractéristiques des trafics analysés et nous présentons dans le tableau 2 les résultats obtenus.

Tab. 1: Caractéristiques du trafic

	Type	Size (Mb)	TCP (%)	UDP (%)	ICMP (%)	Autres (%)	# paquets Fragmentés	# flux TCP
log1	Sim.	$2.10^{-3}$	80	10	10	0	2	1
log2	Sim.	1.2	50	25	25	0	2000	1000
log3	Réel	102	90.1	6.7	3.1	0.1	0	1026
log4	Réel	222	75.3	17.8	6.7	0.2	1	4829
log5	Réel	$10^3$	75.3	17.8	6.7	0.2	46	18460

Il s'avère que notre méthode supporte parfaitement le processus de réassemblage des flux TCP et des fragments IP. Elle présente également des temps nettement meilleurs lorsqu'il s'agit d'un trafic trop fragmenté et des temps comparables en analysant des trafics réels. Par ailleurs, l'algorithme d'Aho Corasick présente souvent de meilleurs résultats. Cela est dû à la taille réduite des motifs.

Tab. 2: Résultats des expérimentations

	Snort 1.9		Snort Modifié			
	Alert	Temps(s)	Alert	FDAWG(s)	AC(s)	MDAWG(s)
log1	2	0.1	3	0.1	<b>0.1</b>	0.1
log2	0	2.4	1	0.11	<b>0.11</b>	0.12
log3	169	9.17	170	9.57	<b>7.62</b>	9.51
log4	1526	30.7	1525	32.39	<b>27.1</b>	31.35
log5	2974	89.76	3001	100.29	<b>83.9</b>	99.79

## 6 Conclusion

Le challenge des IDS actuels est de supporter parfaitement les connexions à haut débit en assurant une détection avec analyse d'états. Dans ce contexte, nous avons proposée dans cet article une méthode en ligne qui réalise ce type de détection. La méthode découvre également les tentatives courantes d'évasion employées pour contourner les équipements de sécurité. Nous envisageons dans de futurs travaux compléter notre méthode de détection par une analyse protocolaire plus approfondie afin de mieux connaître la nature du trafic et par la suite d'optimiser le processus de supervision.

## References

- [CCG<sup>+</sup>93] M. Crochemore, A. Czumaj, L. Gąsieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Fast practical multi-pattern matching. Rapport 93-3, Institut Gaspard Monge, Université de Marne la Vallée, 1993.
- [CL97] M. Crochemore and T. Lecroq. Pattern matching and text compression algorithms. In *Allen B. Tucker, Jr. (Editor-in-Chief), The Computer Science and Engineering Handbook, CRC Press, in cooperation with ACM, 1997*. 1997.
- [Des02] N. Desai. Increasing performance in high speed nids, a look at snort's internals. URL: [www.cis.udel.edu/zhi/www.docshow.net/ids/Increasing\\_Performance\\_in\\_High\\_Speed\\_NIDS.pdf](http://www.cis.udel.edu/zhi/www.docshow.net/ids/Increasing_Performance_in_High_Speed_NIDS.pdf), 2002.
- [JM01] S. Staniford J. McAlerney, C. Coit. Towards faster pattern matching for intrusion detection. DARPA Information Survivability Conference and Exposition (DISCEX II'01), 2001.
- [KNMH00] J. Kuri, G. Navarro, L. Mée, and L. Heye. A pattern matching based filter for audit reduction and fast detection of potential intrusions. In *Proceedings of the 3rd International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, LNCS v. 1907, pages 17–27, 2000.
- [KR97] G. Kucherov and M. Rusinowitch. Matching a set of strings with variable length don't cares. *Theoretical Computer Science*, 178(1–2):129–154, 30 May 1997.
- [KVVK] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. URL:[citeseer.nj.nec.com/kruegel02stateful.html](http://citeseer.nj.nec.com/kruegel02stateful.html).
- [MF01] G. Varghese M. Fisk. Fast content-based packet handling for intrusion detection. Rapport CS 20001-0670, University of California, San Diego, Department of Computer science and Engineering, 2001.
- [MH02] K. Wiley M. Hall. Capacity verification for high speed network intrusion detection systems. In *Proceedings of the 5th International Workshop on the Recent Advances in Intrusion Detection (RAID'2002)*, LNCS v. 2516, pages 239–251, 2002.

- [PCC] Victor K. Wei P. C. Chan. Preemptive distributed intrusion detection using mobile agents. In *Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02)*, page 103.
- [PN98] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, January 1998.
- [RG02] M. Roesch and C. Green. Snort users manual snort release. URL: [www.snort.org/docs/SnortUsersManual.pdf](http://www.snort.org/docs/SnortUsersManual.pdf), 2002.
- [RL00] D.J. Fried J. Korba K. Das R. Lippmann, J. W. Haines. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In *Proceedings of the 3rd International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, LNCS v. 1907, pages 162–182, 2000.
- [Roe99] M. Roesch. Snort — lightweight intrusion detection for networks. In USENIX, editor, *Proceedings of the Thirteenth Systems Administration Conference (LISA XIII): November 7–12, 1999, Seattle, WA, USA, Berkeley, CA, USA, 1999*. USENIX.
- [SGVS99] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag. A high-performance network intrusion detection system. In *ACM Conference on Computer and Communications Security*, pages 8–17, 1999.
- [WDD00] A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In *Proceedings of the 3rd International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, LNCS v. 1907, pages 110–129, 2000.