Détection des anomalies dans les pare-feux de nouvelles générations

Frédéric Cuppens[1], Nora Cuppens-Boulahia[1], Joaquin Garcia-Alfaro[1],
Tarik Moataz[1], Stéphane Morucci[2], Xavier Rimasson[1]

[1]Télécom Bretagne, Campus de Rennes
Cesson-Sévigné, France
Email : {Prenom.Nom}@telecom-bretagne.eu

[2]Swid Web Performance Service
Rennes, France
Email : {Prenom.Nom}@swid.fr

***Résumé*—Une politique de sécurité est constituée d'un ensemble de règles visant à protéger un système d'information. Pour assurer cette protection, les règles doivent être déployées sur les composants de sécurité de manière cohérente et non redondante. Malheureusement, une approche empirique est souvent adoptée par les administrateurs réseau, au détriment d'une validation théorique. Alors que la littérature sur l'analyse des configurations des pare-feux sans état est aujourd'hui riche, ce n'est pas le cas pour les pare-feux à états et applicatifs. Après une description des principales fonctionnalités des pare-feux à états, nous proposons une typologie et une correction des anomalies au sein d'un ensemble de règles à états. Nous terminons par l'analyse des anomalies entre des règles sans état et avec états.**

***Mots clefs*—Politiques de sécurité, configuration de composants réseaux, pare-feux, détection et correction des anomalies.**

I. Introduction

Une politique de sécurité est constituée d'un ensemble de règles, qui une fois spécifiées, doivent être déployées dans le système d'information à protéger. Ce déploiement consiste à distribuer les règles de la politique sur les différents composants de sécurité du système d'information tels que pare-feux, systèmes de détection et de prévention d'intrusion, proxies, etc. Les règles de sécurité doivent être déployées de façon cohérente, non redondante et, dans la mesure du possible, optimale. Les approches reposant sur les techniques de raffinement formel, en utilisant par exemple des machines abstraites spécifiées dans la théorie des ensembles et la logique du premier ordre, permettent de garantir, par construction, la cohésion, la complétude et l'optimisation du déploiement [1].

Malheureusement, le plus souvent, ces approches ne sont pas utilisées par les administrateurs de sécurité. Ces derniers déploient généralement la politique de sécurité de façon manuelle et empirique. Il est alors pertinent d'analyser les règles de sécurité déployées pour détecter et corriger certaines erreurs de déploiement, connues dans la littérature sous le nom *d'anomalies de configuration intra et inter composants* [2].

De nombreux travaux ont déjà été menés pour analyser les anomalies de déploiement [3], [4], [5], [2] et ont abouti à des résultats théoriques et des développements très intéressants pour l'assainissement des configurations. Dans cet article, nous nous intéressons plus particulièrement aux anomalies de configuration dans les pare-feux de nouvelle génération. Les pare-feux [6] sont des composants de sécurité matériel ou logiciel qui contrôlent le trafic entre le réseau public et les zones privées d'un réseau local ainsi qu'entre les zones privées de ce réseau. Le trafic indésirable est bloqué par le pare-feu ou bien re-routé vers d'autres composants pour analyse.

A l'origine, les premiers pare-feux ne permettaient qu'un filtrage *sans état* du trafic réseau. Les pare-feux sans état décident si un paquet est accepté ou refusé, suivant un ensemble de règles de configuration statiques. Ils constituent la solution de filtrage la plus simple. Leur avantage principal est la rapidité des opérations de filtrage. En revanche, ils ne sont pas capables de détecter certaines vulnérabilités sur les paquets car ils n'analysent pas le contenu de ces derniers.

Plus récemment, le concept de pare-feu à états a été développé. Ces pare-feux améliorent le filtrage des paquets en suivant l'état de connexion et en bloquant les paquets ne respectant pas l'automate d'états du protocole au niveau de la couche transport. Comme pour le filtrage simple de paquets, le filtrage à états intercepte les paquets au niveau de la couche réseau et les inspecte afin de vérifier s'ils sont autorisés par une règle de configuration du pare-feu. Cependant, à la différence des pare-feux sans état, les pare-feux à états conservent la trace de chaque connexion dans une table d'états. Bien que les entrées de la table d'états varient selon le constructeur du pare-feu, elles comprennent généralement les adresses IP source et destination, les numéros de port et les informations relatives à l'état de connexion.

Il existe aujourd'hui une troisième catégorie de pare-feux : les pare-feux *applicatifs*. Ces pare-feux effectuent un filtrage au niveau de la couche applicative et permettent ainsi d'ana-

lyser le contenu stocké dans les paquets. Cette fonctionnalité permet de contrôler l'exécution de certaines commandes ou les échanges de données particulières. Par exemple, une règle de configuration d'un pare-feu applicatif pourrait interdire la commande « PUT » du protocole FTP afin d'empêcher le téléchargement sur certaines adresses IP. Le protocole HTTP peut lui aussi être filtré afin de bloquer, par exemple, l'exécution de certaines applications Java. Les pare-feux applicatifs sont cependant incapables d'inspecter des données chiffrées et ils ne supportent pas tous les protocoles applicatifs.

La plupart des méthodes qui ont été proposées pour détecter les anomalies de configuration dans les pare-feux telles que [3], [4], [5], [2] sont limitées aux configurations des pare-feux sans état. La détection des anomalies dans les pare-feux à états ou applicatifs est encore une problématique de recherche peu explorée. Quelques approches ont été proposées dans le cas de pare-feux à états [7] mais les auteurs se sont limités à décrire un modèle alors que dans [8] les auteurs proposent une méthode incomplète car ne considérant que les anomalies déjà définies pour les pare-feux sans état. Dans le cas des pare-feux applicatifs, il n'existe pas à notre connaissance de travaux portant sur ce sujet.

Dans cet article, nous proposons d'étendre les algorithmes de détection d'anomalies définis dans [2] au cas des pare-feux à états. Le principe de la démarche repose sur la spécification d'un automate décrivant les différents états que peuvent prendre les paquets tout au long du processus de filtrage. Nous proposons ensuite une classification des différentes anomalies de configuration dans le cas des pare-feux à états. Nous montrons notamment que les pare-feux à états permettent de combiner des règles de configuration à états et des règles de configuration sans état. Certaines anomalies se produisent entre règles de configuration à états ; nous les appelons anomalies intra règles à états dans la suite. D'autres anomalies concernent les interactions entre règles de configuration à états et sans état ; nous les appelons anomalies inter règles à états et règles sans état. Nous définissons ensuite des algorithmes permettant de détecter automatiquement ces différentes anomalies. Ces algorithmes sont génériques et permettent d'analyser différents protocoles à états, tels que TCP ou FTP, en prenant l'automate d'états du protocole comme paramètre.

L'outil MIRAGE [9] a été implémenté en JAVA et permet la détection automatique des anomalies de configuration mais en se limitant aux pare-feux sans état. Les algorithmes présentés ici fournissent une extension de MIRAGE afin de détecter des anomalies dans le cas des pare-feux à états.

Le reste de l'article est organisé de la façon suivante. La section II revient de façon plus détaillée sur les principales fonctionnalités des pare-feux à états. La section III propose un état de l'art des anomalies dans les pare-feux, en commençant par les pare-feux sans état puis en détaillant les propositions actuelles pour les pare-feux à états. La section IV présente une classification des différents types d'anomalies intra règles de configuration à états, analyse ensuite le cas particulier du protocole TCP et définit un algorithme général pour détecter et corriger les différents types d'anomalies pour n'importe quel protocole à états. La section V procède de même pour la détection des anomalies inter règles à états et règles sans état. La section VI conclut l'article et recense des prolongements possibles pour les travaux présentés dans cet article.

II. Fonctionnalités des pare-feux à états

Le filtrage à états représente une évolution directe du filtrage sans état. En effet, la plupart des paquets circulant sur le réseau utilise le protocole TCP comme protocole de la couche transport. La logique veut que le filtrage évolue vers un filtrage dynamique dépendant des états protocolaires. Ainsi le filtrage ne se basera pas seulement sur les règles de la couche 3 pour accepter un paquet mais également sur l'état du protocole.

Les avantages de ces pare-feux à états sont la granularité du filtrage, l'introduction de l'état comme attribut supplémentaire rendant le filtrage plus fin. D'autre part, ce type de pare-feu permet de conserver et d'analyser les traces de connexions des utilisateurs. Ces pare-feux protègent également contre certains types d'attaques plus compliquées à gérer par des règles sans état notamment le déni de service (DOS) et l'IP Spoofing. Les inconvénients de ce type de filtrage sont une implémentation plus complexe et une utilisation plus importante de la mémoire pour retenir les sessions précédentes.

Si l'on considère des protocoles orientés connexions, TCP, DCCP, ATM, Frame Relay, TIPC, SCTP, IPX/SPX, etc., tous ces protocoles reposent sur un principe commun : l'envoi d'un flux tagué appartenant à une session déterminée. Dans le cas de TCP, cette session s'appelle une connexion, pour SCTP il s'agit plutôt d'une association. Dans la littérature, il n'existe pas de modèle formel qui décrit un comportement général des pare-feux à états avec toutes les spécifications nécessaires et une couverture de tous les protocoles. Gouda et Liu [7] ont proposé dans leur travaux une modélisation d'une inspection à état, associant cette inspection à une juxtaposition des phases de détections *à états* puis *sans état*. Ils ont proposé l'ajout de l'attribut *état* dépendant du protocole utilisé permettant de stocker des paquets appartenant à l'initialisation de la connexion, et un *tag* permettant de vérifier l'appartenance des nouveaux paquets à cette connexion.

Dans cet article l'attribut état représente un état du protocole lui-même, ce qui n'implique pas nécessairement l'initialisation d'une session orientée connexion (par exemple le protocole ICMP, peut être modélisé par des états résultant des échanges de messages echo-request et echo-reply, même si ce protocole n'est pas orienté connexion). Cette approche correspond au modèle proposé pour les pare-feux à états, sauf que nous pouvons proposer d'autres modèles vérifiant la même fonctionnalité, comme l'existence simultanée des règles à états et sans état. Plus précisément, au lieu de proposer un filtrage en série passant par *l'état* puis par les attributs présentés dans le modèle sans état, nous considérons une modélisation du filtrage intégrant des règles à états et sans état en parallèle.

Indépendamment de la modélisation, l'objectif des pare-feux à états est de filtrer les paquets selon l'appartenance ou non à une connexion/session. Ainsi selon la définition de la session, de la modélisation et de la représentation des

règles dans les tables, la spécification et la détermination des anomalies seraient différentes. Dans cet article, nous nous intéressons, dans un premier temps, à la détection d'anomalie dans les pare-feux à états ne contenant que des règles à état. Plus précisément, nous allons définir et analyser de nouveaux types d'anomalies dépendant des transitions du protocole considéré. Le protocole choisi pour illustrer l'approche est TCP en tant que référence de protocole orienté connexion.

III. Anomalies dans les pare-feux

Les travaux portant sur la conception de pare-feux, essentiellement sans état pour la plupart, se sont intéressés aux langages de haut niveau pour la spécification des règles de filtrage. Certains de ces langages sont plutôt fonctionnels tels que celui proposé par Guttman [10], d'autres se fondent sur un modèle de sécurité simple d'expression des règles comme celui défini par Bartal *et al.* [11] ou s'appuient sur des modèles plus riches permettant de capturer d'autres aspects liés au filtrage comme la topologie du réseau à sécuriser (par exemple l'existence de plusieurs VLAN ou le déploiement de plusieurs pare-feux) [12]. L'utilisation de tels langages ou modèles d'expression de politiques de sécurité réseau permet lors de l'écriture des règles de s'affranchir de la syntaxe souvent très spécifique des différents types de firewall proposés sur le marché et, pour certains d'entre-eux, de les déployer de façon automatique. Les recherches et développements dans ce contexte, notamment ceux qui proposent des modèles plus évolués d'expression et de déploiement de règles de sécurité réseau, permettent de vérifier la satisfaisabilité ou de restaurer des propriétés de cohérence (absence de conflits), de complétude (l'ensemble des exigences cibles est couvert) ou de compacité (pas de règles redondantes ou inutiles) [1]. Ces approches descendantes peuvent également tenir compte des fonctionnalités offertes par ces pare-feux (à états ou sans état, gestion de réseaux privés virtuels, etc.) [13] pour assurer une répartition efficace des tâches entre le module de décision et le composant de filtrage.

Mais, dans le cas de configurations de pare-feux déjà déployées, ces travaux ne résolvent pas les problèmes de conflits résultant de permissions et d'interdictions s'appliquant aux mêmes paquets, de redondance de règles souvent inutiles ou d'incomplétude due, par exemple, à la présence de règles masquées. Plusieurs travaux ont été menés pour analyser les anomalies de configurations existantes. Nous pouvons les classer dans trois catégories : (I) ceux orientés interrogation du pare-feu [14], [15] ou [16], (II) ceux ciblant la gestion de conflits [17], [18] et (III) ceux se focalisant sur la détection des anomalies [19], [4], [2], [20], [21]. Dans la catégorie I, la problématique d'analyse a été réorientée vers un processus de recherche d'information en interrogeant le pare-feu, ce qui déplace la difficulté vers la structuration des configurations, le langage de requêtes, l'exhaustivité et l'efficacité de ces requêtes, le besoin de mettre en vis-à-vis la politique souhaitée et les règles de filtrage ce qui rend l'analyse complexe, sujette aussi bien à des faux positifs qu'à des faux négatifs puisque la dimension histoire n'est pas prise en compte. La catégorie II s'intéresse aux algorithmes de classification de paquets plutôt dans les routeurs, tels que Gried-of-tries [22] ou ABV [17], par la définition de structures de données permettant de trouver rapidement les règles correspondant à un paquet entrant puis détecter si elles ne préconisent pas des actions contraires. Enfin, la catégorie III est proche de la catégorie II mais se veut plus précise et plus enveloppante en (1) caractérisant les erreurs de configuration qui vont au-delà du conflit, par exemple, les règles redondantes sont aussi considérées, en (2) s'affranchissant de l'ordre des règles, en (3) ne traitant pas les règles deux à deux comme le fait incorrectement Al-Shaer *et al.* [19] ce qui permet de détecter par exemple des masquages partiaux [2] et en (4) étendant la vision locale de la détection des erreurs aux liens inter pare-feux des configurations qui peut induire d'autres anomalies comme la "mis-connection", c'est-à-dire un paquet autorisé qui ne peut atteindre sa destination.

Ces travaux d'analyse de configurations déployées ne considèrent pas pour la plupart le cas des pare-feux avec états. On constate cependant quelques études comme celles de Buttyán *et al.* [8], mais l'approche est incorrecte puisqu'elle essaye de ramener la problématique de détection des anomalies à celle utilisée dans le cas des pare-feux sans état. Par conséquent, ces travaux ne tiennent pas compte des anomalies liées à une mauvaise gestion de l'histoire et de la mémoire gérée par la table d'états. En utilisant une approche se fondant sur le web sémantique, Fitzgerald *et al.* [23], modélisent des pare-feux aussi bien avec que sans état. Cette représentation est intéressante puisqu'elle est générique, mais l'analyse des erreurs proposée ne caractérise pas les types d'erreurs qui permettraient leur traitement, et elle se fonde essentiellement sur les "bonnes" pratiques en termes de définition de règles de configuration de pare-feux, ce qui augmente le taux de faux négatifs.

IV. Détection et correction des anomalies dans les règles à états d'un pare-feu

A. Classification des anomalies intra règles

Il s'agit d'identifier des anomalies pour les règles à états des pare-feux. Les anomalies étudiées se concentrent sur la couche transport de la pile protocolaire du modèle OSI.

Pour un protocole de cette couche, il faut distinguer trois phases importantes lors d'une communication entre un serveur et un client. Premièrement, la phase de connexion ; par exemple, dans le cas de TCP :

- Le client envoie SYN (un paquet avec le drapeau (*flag*) SYN) ;
- Le serveur renvoie SYN+ACK ;
- Le client répond avec ACK.

Ensuite vient une phase de transfert de données. Enfin, la communication se termine par une phase de fermeture. Si c'est le client qui demande la fermeture, on a le scénario suivant (toujours dans le cas de TCP) :

- Le client envoie FIN (un paquet avec le drapeau FIN) ;
- Le serveur répond avec ACK puis FIN ;
- Le client répond avec ACK.

Dans le cas des pare-feux sans état, nous considérons les types d'anomalies suivants :

- Masquage (*shadowing*) : la règle R_i est masquée dans l'ensemble des règles R du pare-feu si et seulement si elle n'est jamais appliquée car tous les paquets qu'elle devrait traiter sont interceptés par une ou plusieurs règles placées en amont ;
- Redondance : la règle R_i est redondante dans l'ensemble des règles R du pare-feu si et seulement si (1) elle n'est pas masquée par une ou plusieurs autres règles et (2) si sa suppression ne change pas le filtrage final opéré par le pare-feu.

Les pare-feux à états sont aussi concernés par ces anomalies rencontrées dans le cas sans état ; les définitions restent les mêmes.

Nous avons, par ailleurs, identifié un autre type d'anomalie concernant les règles à états, dites anomalies *protocolaires*, dans le sens où un ensemble de règles présentant ce défaut risque de compromettre le bon déroulement d'une communication d'un protocole tel que TCP. Les anomalies que nous avons détectées interviennent au niveau de l'établissement d'une connexion ainsi que lors de la fermeture. En effet, nous pouvons considérer deux situations :

1) Le client entame la phase d'établissement d'une connexion acceptée par le serveur, mais le pare-feu interdit l'acquittement de cette ouverture de connexion ;
2) Situation similaire, mais qui se produit au niveau de la fermeture de connexion.

Dans la suite, nous allons présenter des algorithmes pour traiter et éliminer ces anomalies lors de l'établissement et la fermeture de connexion.

B. Élimination des anomalies protocolaires

1) Définitions : La communication entre deux hôtes avec un protocole de la couche transport se compose d'une phase de connexion et d'une phase de fermeture, une fois le transfert terminé. Considérons pour un protocole de cette couche deux automates finis déterministes A_1 et A_2, modélisant respectivement le processus d'établissement et de fin de connexion. Nous distinguons deux automates afin d'opérer une correction différente sur les règles présentant des anomalies, selon qu'elles interviennent sur le processus d'ouverture ou de fermeture de connexion. Nous justifierons les cas de correction plus loin. Caractérisons ces automates et donnons quelques exemples pour TCP et SCTP (Stream Control Transmission Protocol) :

- Le vocabulaire Σ, l'ensemble des *flags* du protocole ; ainsi, pour TCP :
  $\Sigma = \{SYN, SYN{+}ACK, ACK, FIN, FIN{+}ACK\}$;
  et pour SCTP :
  $\Sigma = \{INIT, ABORT, ASSOCIATE, ...\}$.
- Q, l'ensemble des états du protocole, tels que CLOSED, LISTEN, SYN SENT, ... pour TCP, ou bien COOKIE-WAIT, COOKIE-ECHOED, ... pour SCTP ;
- La fonction de transition $\delta : Q \times \Sigma \rightarrow Q$;

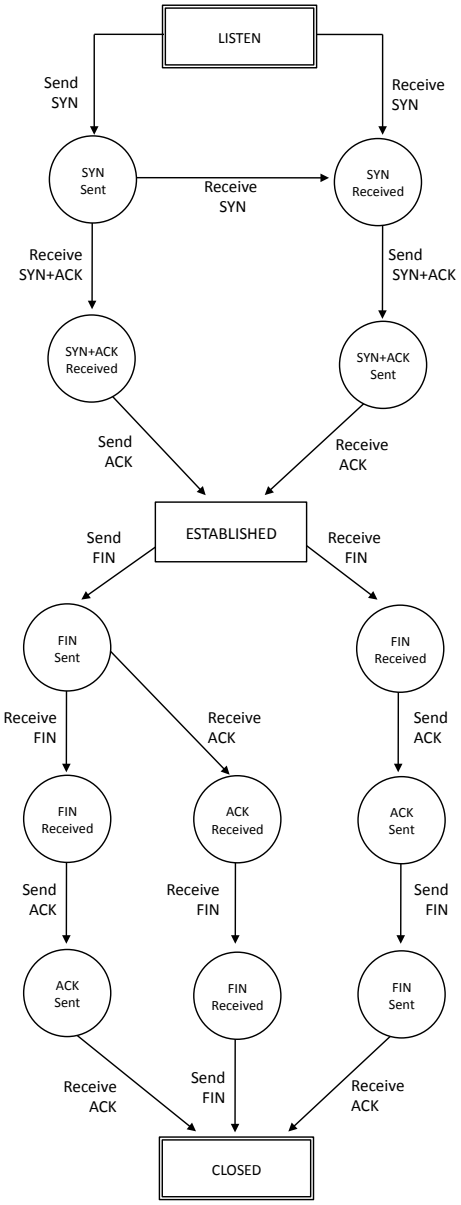

FIG. 1. Automate d'états pour TCP.

- Un état de départ q_0 ; pour TCP, il s'agit de LISTEN pour A_1 et ESTABLISHED pour A_2 ; dans le cas de SCTP, CLOSED pour A_1 et ESTABLISHED pour A_2 ;
- Un état final noté q_f ; pour TCP, celui de A_1 est ESTABLISHED, celui de A_2 CLOSED ; pour SCTP, ESTABLISHED pour A_1 et CLOSED pour A_2.

Nous considérons les fonctions suivantes pour définir les algorithmes :

- etatsAdjacents(A, $Etat_i$, $Etat_j$) : fonction booléenne ; l'expression est vraie si et seulement s'il existe dans l'automate A une transition pour aller de l'état $Etat_i$ à l'état $Etat_j$. De façon équivalente, l'expression est vraie si et seulement s'il existe $a \in \Sigma$ tel que $\delta(Etat_i, a) = Etat_j$
- etatSuivant(A, $Etat$, $Symbole$) : s'il existe, retourne

AddrSource	AddrDest	PortSrc	PortDest	Prot.	Flag	Etat	Decision
41.1.1.1	193.1.1.2	8080	2011	TCP	SYN	LISTEN	~~ACCEPT~~ DENY
193.1.1.2	41.1.1.1	2011	8080	TCP	SYN+ACK	SYN RCVD	DENY

FIG. 2. Exemple de correction d'une anomalie protocolaire dans un sous-ensemble de règles à états.

sous forme d'un singleton l'état $EtatSuivant$ tel que $\delta(Etat, Symbole) = EtatSuivant$. Sinon, la fonction renvoie $\emptyset$. Cette définition va nous permettre de simplifier les notations dans les algorithmes.

Enfin, nous considérons un ensemble R de taille n, contenant des règles R_i ($i \leq n$). Chaque règle R_i est modélisée par ces caractéristiques :

- $R_i[addrDestination]$: adresses destination ;
- $R_i[portSource]$: ports source ;
- $R_i[portDestination]$: ports destination ;
- $R_i[Protocole]$: protocole ;
- $R_i[Flag]$: flag ;
- $R_i[Etat]$: état du protocole.

La figure 1 illustre l'automate du protocole TCP. Il comprend les automates A_1 et A_2, qui partagent en commun l'état ESTABLISHED.

2) Algorithmes : On suppose qu'on traite un sous-ensemble R de règles R_i dont le protocole (attribut $R_i[protocole]$) correspond à un protocole donné.

Ces algorithmes se fondent sur les automates d'établissement et de fermeture de connexion du protocole fixé. On les appellera respectivement A_1 et A_2.

Si un couple de règles concerne deux états adjacents de l'automate, on vérifie si les *flags* associés sont bien les symboles qui permettent de passer d'un de ces états à l'autre. Si c'est le cas, et que les décisions des deux règles sont différentes, on caractérise une anomalie.

On applique l'algorithme de correction des anomalies d'abord sur les règles relatives à l'établissement de connexion, puis à celles de fin de connexion. Pour les anomalies sur l'établissement de connexion, la correction consiste à positionner les décisions des règles sur DENY : on évite ainsi les échecs de connexion. Pour les anomalies sur la fermeture de connexion, on positionne les décisions des règles par défaut sur ACCEPT, afin d'éviter les échecs de fermeture de connexion.

L'algorithme 1 rassemble à un niveau macroscopique les corrections apportées ; il utilise l'algorithme 2 qui analyse les règles, détecte ce type d'anomalies et effectue ces corrections.

La figure 2 donne un exemple de correction d'un sous-ensemble de règles à états avec l'algorithme. Dans la configuration initiale de l'exemple, une première règle accepte les paquets avec le flag SYN et l'état LISTEN. Si on se réfère à l'automate TCP de la figure 1, la règle autorise la première étape d'une connexion TCP sur un serveur (passage de l'état LISTEN à l'état SYN *Received*). Mais la deuxième règle interdit l'acquittement en retour. La connexion ne pourra donc pas aboutir. Dans ce cas, l'algorithme va modifier la première règle, de façon à refuser la connexion.

V. DÉTECTION ET CORRECTION DES ANOMALIES ENTRE RÈGLES À ÉTATS ET RÈGLES SANS ÉTAT

A. Concepts et exemples

A ce point de l'article, nous avons étudié des anomalies dans un ensemble de règles à états, au niveau de la couche transport. De fait, la mixité de règles avec et sans état est envisageable dans une configuration d'un pare-feu. Par exemple, un administrateur réseau peut rajouter une règle pour gérer les connexions TCP utilisées pour le transfert des données dans le cadre d'une session FTP.

Algorithme 1: `AnomaliesProtocole`(R, A_1, A_2)

```
/*Correction pour l'automate de connexion */
R ← correction(R, A1, DENY)
/*Correction pour l'automate de fermeture */
R ← correction(R, A2, ACCEPT) retourner R
```

Algorithme 2: `correction`($R, Automate, Decision$)

```
n ← size(R) ;
pour i ← 1 à n − 1 faire
    pour j ← i + 1 à n faire
        si Ri[addrSource] = Rj[addrDestination]
           ∧ Ri[portSource] = Rj[portDestination]
           ∧ Ri[decision] ≠ Rj[decision] alors
            si etatsAdjacents (Automate, Ri[etat],
            Rj[etat]) alors
                si (Rj[etat] ∈ etatSuivant (Automate,
                Ri[etat], Ri[flag])
                   ∧ etatSuivant (Automate, Rj[etat],
                Rj[flag]) ≠ ∅)
                ∨
                (Ri[etat] ∈ etatSuivant (Automate,
                Rj[etat], Rj[flag])
                   ∧ etatSuivant (Automate, Ri[etat],
                Ri[flag]) ≠ ∅) alors
                    Ri[decision] ← Decision
                    Rj[decision] ← Decision
                fin
            fin
        fin
    fin
fin
retourner R
```

Dans un pare-feu Netfilter, cela se traduirait par l'ajout d'une règle avec l'état RELATED. La règle cohabitera alors avec les autres règles sans état définies par l'administrateur.

A partir de ce constat, nous considérons que la recherche d'anomalies *entre* les règles avec états et celles sans état s'avère pertinente lorsque l'analyse se base sur la spécification d'un protocole de la couche applicative. En effet, un certain

nombre de protocoles applicatifs utilisent plusieurs connexions de type TCP (ou d'un autre protocole de la couche transport) lors d'une session entre deux hôtes. C'est le cas de FTP, d'IRC ou des protocoles de VoIP par exemple, qui utilisent des connexions *annexes* si cela est nécessaire.

Détaillons le cas de FTP. Une session classique avec ce protocole se déroule en deux étapes :

1) Le client contacte le serveur FTP sur son port 21, et une connexion TCP – dite *de contrôle* – est établie ;
2) Lorsque le client souhaite effectuer un transfert de données (envoi d'un fichier, récupération d'une liste de répertoire, etc.), deux cas possibles se présentent :
   – Après négociation de la connexion de contrôle, le serveur initie une nouvelle connexion TCP pour le transfert depuis le port 20 vers le client qui aura pris soin d'ouvrir le port sur lequel le serveur sera contacté pour recevoir les données. C'est le mode *actif* ;
   – Ou bien, c'est le client qui initie la nouvelle connexion vers le serveur, qui aura renseigné au client le port à utiliser. C'est le mode *passif*.

La configuration du pare-feu du serveur FTP doit contenir :
– Une règle sans état pour autoriser les paquets TCP à destination du port 21 ;
– Une règle à états pour autoriser les paquets *dont la connexion TCP associée a été identifiée comme une connexion annexe dans une session FTP*. Le port sera, selon les cas, soit 20 (mode actif), soit un port supérieur à 1024 (mode passif).

A travers cet exemple, nous souhaitons souligner qu'il s'agit d'assurer une gestion correcte des connexions TCP annexes entre deux hôtes utilisant un protocole applicatif. Notons que le pare-feu doit avoir une connaissance du protocole applicatif concerné par les règles, afin de décider si un paquet fait partie d'une connexion annexe ou non. En particulier, Netfilter possède un module pour suivre les sessions FTP. Ainsi, un paquet d'une connexion TCP de données (au sens de FTP) est marqué avec le statut RELATED. Cette marque du contexte au niveau applicatif offre alors à l'administrateur la possibilité de définir des règles avec états.

B. Algorithme de détection d'anomalies

Afin de parvenir à une découverte systématique de telles anomalies pour un protocole applicatif donné, nous supposons disposer d'une spécification exhaustive des scénarios possibles en termes de connexions TCP pour ce protocole lors d'une session entre deux hôtes. Cette spécification décrit comment débuter la connexion, et comment les connexions annexes interviennent dans la session (ordre, nombre, ports, etc.).

La première étape consiste à chercher des règles sans état qui correspondent à l'établissement de la connexion pour ce protocole. Par exemple, pour FTP, on recherche une règle concernant les paquets TCP à destination du port 21. Si de telles règles sont présentes, nous envisageons les trois cas suivants :

1) Il existe des règles à états pour traiter les connexions annexes succeptibles d'être utilisées par ce protocole ;
2) Il existe des règles sans état pour traiter ces connexions annexes ;
3) Il n'existe aucune règle pour traiter les connexions annexes.

Le cas 2 est trop général, car il ne prend pas en compte la logique du protocole. Un attaquant serait capable d'ouvrir une connexion TCP sur un port qui ne doit être utilisé que dans le cadre d'une session applicative. Dans le cas de FTP, une connexion sur le serveur vers le port *p* ne devrait être autorisée que si le serveur a convenu avec le client d'un transfert FTP en mode passif sur le port *p* du serveur. Dans le cas 3, la session applicative risque de connaître un dysfonctionnement car le pare-feu interdira probablement les connexions annexes. Enfin, le cas 1 comble les défauts des autres situations et correspond à la politique souhaitée par l'administrateur du réseau. Dans Netfilter, de telles règles pour FTP auraient l'état RELATED.

Typiquement, un algorithme de détection de ces anomalies doit signaler à l'administrateur réseau les cas 2 et 3 pour les protocoles applicatifs gérés par les règles.

C. Algorithme de détection

Pour établir un algorithme de détection des anomalies entre règles avec états et celles sans état, posons quelques définitions :

– L : ensemble des règles sans état (*stateLess*), composé de règles L_i (i entier naturel) caractérisées par les attributs $L_i[addrSource]$, $L_i[addrDestination]$, $L_i[portSource]$, $L_i[portDestination]$ et $L_i[Protocole]$ (TCP, UDP, ou autre protocole de la couche transport), tels qu'ils ont été définis dans la section précédente.
– F : ensemble des règles avec états (*stateFul*), composé de règles F_i, caractérisées par les mêmes attributs que les règles L_i, auxquels on ajoute l'attribut $F_i[Etat]$ pour l'état. Il est important de considérer $F_i[Protocole]$ car le protocole transport d'une connexion annexe peut différer de celui de la connexion principale (en VoIP, des données peuvent être transmises via UDP, tandis que la connexion principale est en TCP).
– A : un automate déterministe à nombre fini d'états pour décrire un protocole applicatif. Nous proposons d'identifier le vocabulaire Σ de A comme l'ensemble des commandes pouvant être échangées par les hôtes avec ce protocole, typiquement sur la connexion principale. Ensuite, dans l'ensemble Q des états, nous identifions un sous-ensemble Q_2. Les éléments q de Q_2 symbolisent des créations de connexions TCP (ou tout autre protocole de cette couche) annexes. On peut accéder aux mêmes attributs que les règles sans état sur ces éléments ($q[addrSource]$ adresse source de la connexion, $q[addrDestination]$ adresse destination, etc.). Noter que $q[Etat]$ est très dépendant de l'implémentation du pare-feu, aussi nous définissons dans la prochaine définition une fonction qui associe à un état q de l'automate la représentation de cet état pour le pare-feu considéré. Lorsqu'on parlera de $R_i[Etat]$ (l'état spécifié dans la règle R_i), il s'agira alors d'un état tel que se le représente

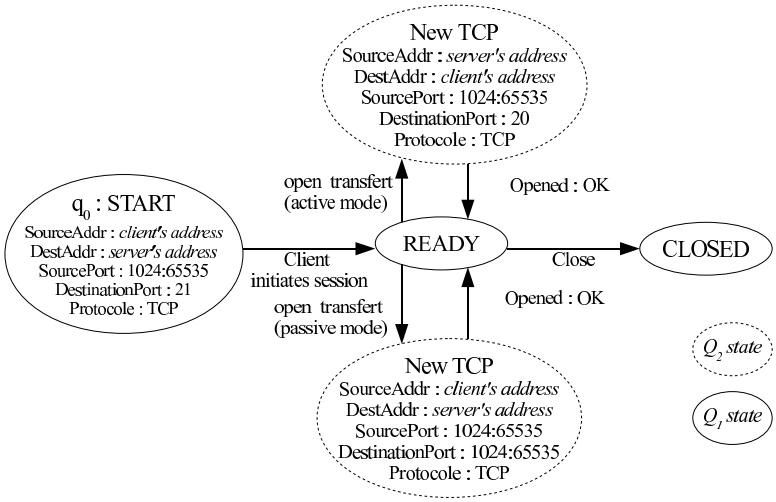

FIG. 3. Proposition d'un automate d'états du protocole applicatif FTP.

le pare-feu, et non pas directement un état de l'automate. Dans Netfilter, les connexions annexes sont identifiées avec l'état RELATED. On pourrait imaginer une extension du pare-feu pour gérer plus finement les états, pour un protocole applicatif donné. On pose $Q_1 = Q - Q_2$, l'ensemble des états qui n'appellent pas à la création de connexions annexes. Pour ces états, $q_[Etat]$ n'est pas défini. Enfin, à l'état initial q_0 de l'automate (qu'on supposera unique), on associe les attributs de ports $q_0[portSource]$, $q_0[portDestination]$ et de protocole de la couche transport $q_0[Protocole]$. La figure 3 donne un exemple d'automate pour FTP.

- $etatParefeu(q)$: fonction qui associe à un état $q \in Q_2$ de l'automate l'état correspondant pour le pare-feu considéré. Par exemple, pour le protocole FTP et le pare-feu Netfilter, la fonction renvoie RELATED pour les deux états qui appellent à la création d'une connexion.
- $existeRegle(R, q)$: fonction booléenne. R est un ensemble de règles avec états ou sans état (l'un ou l'autre, mais pas les deux), q est un état de l'automate A appartenant à Q_2 (états qui symbolisent des créations de connexions annexes). Dans le cas où R est un ensemble de règles sans état, $existeRegle(R, q)$ est vrai si et seulement s'il existe une règle $R_i \in R$ telle que :
  - $q[addrSource] \in R_i[addrSource]$,
  - $q[addrDestination] \in R_i[addrDestination]$,
  - $q[portSource] \in R_i[portSource]$,
  - $q[portDestination] \in R_i[portDestination]$,
  - $q[protocole] = R_i[protocole]$.

  Dans le cas où R est un ensemble de règles avec états, l'expression est vraie si toutes les conditions précédentes sont réunies, ainsi que la condition $etatParefeu(q[Etat]) = R_i[etat]$.
- $existeRegle(L, q_0)$: fonction booléenne. q_0 est l'état initial du protocole et L un ensemble de règles sans état. Cette fonction est vraie si et seulement s'il existe une règle $L_i \in L$ telle que :
  - $q_0[portSource] \in L_i[portSource]$,
  - $q_0[portDestination] \in L_i[portDestination]$,
  - $q_0[Protocole] = L_i[Protocole]$.

L'algorithme 3 va consister à étudier chaque état de Q_2 de l'automate d'un protocole applicatif donné, en cherchant des règles qui correspondent aux connexions annexes associées. On affiche un message selon ce qu'on trouve, conformément aux trois cas énumérés dans la sous-section précédente (absence, présence dans la partie avec états, présence dans la partie sans état). $A[Q_2]$ désigne l'ensemble Q_2 de l'automate.

L'algorithme 4 permet de détecter des anomalies entre les règles avec états et sans état, étant donnée une « bibliothèque » de protocoles applicatifs dont on connaît les automates. Pour chaque protocole applicatif, nous vérifions si le pare-feu l'autorise grâce aux attributs de l'état initial q_0 de l'automate du protocole. Si c'est le cas, on utilise l'algorithme 3 pour détecter des anomalies concernant ce protocole. $A[q_0]$ désigne l'état initial q_0 de l'automate.

Algorithme 3: `anomaliesInter`(L, F, A)

```
/*A[Q2] : états de Q2 pour l'automate A */
pour tous les q ∈ A[Q2] faire
    si existeRegle(F, q) alors
        /*Passage à l'état suivant */
        suivant ;
    fin
    si existeRegle(L, q) alors
        afficher("Attention : règle sans état pour l'état q
        du protocole A")
    sinon
        afficher("Attention : pas de règle pour l'état q du
        protocole A")
    fin
fin
```

La figure 4 est un extrait de configuration d'un pare-feu Netfilter, configuré pour autoriser l'accès à un serveur FTP,

Algorithme 4: `anomaliesInterTousPro`(L, F, Bib)

```
/*Bib : bibliothèque des automates des
  protocoles applicatifs supportés */
pour tous les A ∈ Bib faire
    si existeRegle(L, A[q0]) alors
        anomaliesInter(L, F, A)
    fin
fin
```

en mode actif et passif. Cette configuration présente une anomalie entre règles à états et sans état. R_1 est une règle sans état qui autorise l'établissement d'une connexion FTP. Pour gérer les connexions de transfert, les règles R_2 et R_3 ont été rajoutées. Cependant, elles sont sans état, et donc leur champ d'application est trop large. L'algorithme 4 signalera ces deux règles. Pour corriger ce défaut, l'administrateur doit ajouter l'option `--state RELATED` pour ces deux règles.

R_1: iptables –A FORWARD -s $ANY -d $SERVERS –sport 1024:65535 –dport 21 -j ACCEPT

...

R_2: iptables –A FORWARD -d $ANY -s $SERVERS –dport 1024:65535 –sport 20 -j ACCEPT

R_3: iptables –A FORWARD -s $ANY -d $SERVERS –sport 1024:65535 –dport 1024:65535 -j ACCEPT

FIG. 4. Exemple de configuration FTP sur un pare-feu Netfilter.

VI. CONCLUSION

Au terme de cet article, nous avons caractérisé des types d'anomalies pouvant survenir au sein d'un ensemble de règles à états d'un pare-feu. De plus, nous disposons d'un algorithme pour détecter et modifier les règles présentant des anomalies, afin d'éviter les échecs de connexion et de fermeture de session. Enfin, nous avons vu que l'étude des anomalies entre règles avec états et sans état peut être pertinente lorsqu'elle est mise en perspective avec un protocole de la couche applicative. Détecter de telles anomalies contribue à rationaliser la gestion des connexions TCP annexes requises par certains protocoles de la couche applicative. Concernant l'analyse des anomalies dans les règles à états, un prolongement possible consisterait à formaliser une notion d'équivalence entre l'ensemble des règles avant correction et celui après avoir appliqué l'algorithme de correction. Pour terminer, l'identification des anomalies entre règles à états et sans état que nous proposons s'articule autour de la notion de connexion annexe. Des travaux sont en cours pour développer cette idée, afin d'identifier d'autres scénarios où l'analyse des anomalies entre ces deux ensembles de règles s'avère pertinente.

Acknowledgements : This research was partially supported by the European Commission, in the framework of the ITEA2 Predykot project (Grant agreement no. 10035).

RÉFÉRENCES

[1] S. Preda, N. Cuppens-Boulahia, F. Cuppens, J. Garcia-Alfaro, and L. Toutain, "Model-Driven Security Policy Deployment : Property Oriented Approach," in *ESSoS*, Pisa, Italy, 2010, pp. 123–139.

[2] J. Garcia-Alfaro, N. Boulahia-Cuppens, and F. Cuppens, "Complete analysis of configuration rules to guarantee reliable network security policies," *Int. J. Inf. Sec.*, vol. 7, no. 2, pp. 103–122, 2008.

[3] H. Adiseshu, S. Suri, and G. Parulkar, "Detecting and Resolving Packet Filter Conflicts," in *INFOCOM*, Tel Aviv, Israel, 2000, pp. 1203–1212.

[4] E. Al-Shaer and H. Hamed, "Discovery of Policy Anomalies in Distributed Firewalls," in *INFOCOM*, Hong Kong, China, 2004.

[5] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, and P. Mohapatra, "FIREMAN : A Toolkit for FIREwall Modeling and ANalysis," in *IEEE Symposium on Security and Privacy*, Berkeley, California, USA, 2006, pp. 199–213.

[6] W. Cheswick, S. Bellovin, and A. Rubin, *Firewalls and Internet Security : Repelling the Wily Hacker, 2nd edn.* Addison-Wesley, 2003.

[7] M. Gouda and A. Liu, "A model of stateful firewalls and its properties," in *DSN*, Yokohama, Japan, 2005, pp. 128–137.

[8] L. Buttyan, G. Pék, and T. V. Thong, "Consistency verification of stateful firewalls is not harder than the stateless case," *Infocommunications Journal*, vol. LXIV, 2009.

[9] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda, "MIRAGE : A Management Tool for the Analysis and Deployment of Network Security Policies," in *DPM/SETOP*, Athens, Greece, 2010, pp. 203–215.

[10] J. Guttman, "Filtering postures : Local enforcement for global policies," in *In Proceedings, 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997, pp. 120–129.

[11] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato : A novel firewall management toolkit," 1999.

[12] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, "A formal approach to specify and deploy a network security policy," in *Formal Aspects in Security and Trust*, 2004, pp. 203–218.

[13] S. Preda, F. Cuppens, N. Cuppens-Boulahia, J. Garcia-Alfaro, and L. Toutain, "Dynamic deployment of context-aware access control policies for constrained security devices," *Journal of Systems and Software*, vol. 84, no. 7, pp. 1144–1159, 2011.

[14] S. Hazelhurst, A. Attar, and R. Sinnappan, "Algorithms for improving the dependability of firewall and filter rule lists," in *DSN*, 2000, pp. 576–585.

[15] A. Liu, M. Gouda, H. Ma, and A. Ngu, "Firewall queries," in *In Proceedings of the 8th International Conference on Principles of Distributed Systems, LNCS 3544, T. Higashino Ed.* Springer-Verlag, 2004, pp. 124–139.

[16] A. Mayer, A. Wool, and E. Ziskind, "Fang : A firewall analysis engine," in *IEEE Symposium on Security and Privacy*, 2000, pp. 177–187.

[17] F. Baboescu and G. Varghese, "Scalable packet classification," in *In ACM SIGCOMM*, 2001, pp. 199–210.

[18] D. Eppstein and S. Muthukrishnan, "Internet packet filter management and rectangle geometry," 2001, pp. 827–835.

[19] E. Al-Shaer and H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *Integrated Network Management*, 2003, pp. 17–30.

[20] J. Garcia-Alfaro, F. Cuppens, and N. Cuppens-Boulahia, "Analysis of policy anomalies on distributed network security setups," in *ESORICS*, 2006, pp. 496–511.

[21] ——, "Management of exceptions on access control policies," in *SEC*, 2007, pp. 97–108.

[22] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *In Proc. of SIGCOMM*, 1999, pp. 135–146.

[23] W. Fitzgerald, S. Foley, and M. Ó. Foghlú, "Network access control interoperation using semantic web techniques," in *WOSIS*, 2008, pp. 26–37.