N° d'ordre: 00000

THÈSE

présentée devant

l'Institut National des Sciences Appliquées de Rennes

pour obtenir le grade de :

Docteur

Mention Informatique

par

Elvis Tombini

Titre de la thèse :

Amélioration du diagnostic en détection d'intrusions : étude et application d'une combinaison de méthodes comportementale et par scénarios

À soutenir le 14 septembre 2006 devant la commission d'examen

Rapporteurs	**S. Benferhat**	Professeur des Universités	Université d'Artois
	B. Le Charlier	Professeur des Universités	Université catholique de Louvain
Examinateurs	**H. Debar**	Expert	France Télécom R&D
	M. Ducassé	Professeur des Universités	INSA de Rennes
	D. Herman	Professeur des Universités	Université de Rennes 1
	L. Mé	Professeur	Supélec

Remerciements

Nanar, comédie potache, film d'auteur ou super production hollywoodienne, il faut en passer par le jugement du jury. A ce titre, je remercie Salem Benferhat et Baudouin Le Charlier d'avoir rapporté cette thèse, dans des délais extrêmements courts. Je remercie aussi Daniel Herman d'avoir accepté de participer à ce jury.

Je veux aussi remercier ceux qui ont activement participé et contribué à la réalisation, la production et à la mise en scène. Hervé Debar pour m'avoir soumis un synopsis prometteur, laissé le crayon et montré le maniement de la caméra. Mireille Ducassé pour avoir su me ~~torturer~~ me guider et corriger mon style durant l'écriture des diverses scènes du scénario. Ludovic Mé pour sa participation active lors des phases de tournage. A vous trois (qui a dit Cerbère ?), vous avez su manier la carotte et les coups de ~~bâtons~~ stylos rouges avec *maestria*.

Toute l'écriture et la réalisation n'auraient pu se faire sans une équipe technique digne de ce nom. Quelques scènes extérieures ont été tournées à Supélec. Je remercie les personnes que j'ai rencontré là-bas pour leur accueil. Mais la grande majorité des plans ont été tournés dans le laboratoire NSS des studios France Télécom R&D. Je remercie bien sûr l'ensemble des personnes que j'ai pu y cotoyer durant ces quatre années, surtout pour toutes les soirées, enfin, les discussions de travail qui ont débordées en dehors des heures de bureaux. Plus particulièrement je remercie Sébastien, Vincent, Bruno, et la petite dernière, Valérie, pour avoir su égayer toutes les scènes tournées malgré une météo incertaine et des conditions parfois difficiles. Un clin d'oeil spécial à Nicolas, stagiaire Jedi, qui a réussi à me mettre le nez dans les stats et en est ressorti indemne (moi pas). Je remercie aussi, avec une mention particulière, Hervé et Emmanuel. Finalement, je ne saurais oublier de remercier chaleureusement celui qui a partagé toutes mes bandes sons durant plus de trois années, Benoît (auteur méconnu du fameux "pin ouin, pin ouiiiiin").

Il y a aussi tous ceux qui m'ont accompagné jusqu'au dernier "clap". Peu importe le nombre d'entrées dans les salles obscures, grâce à eux, le *making of* est plus qu'un succès. Chacun a eu son rôle, grand ou très grand (le rôle, pas leur taille, parce qu'Aline elle est un peu petite). Pour tout plein de choses et par ordre alphabétique, merci à Aline (même s'il y a toujours pire ailleurs, je n'ai pas trouvé mieux), Batz (le garçon qui fait Boum ! Boum ! Boum !), Benjamin (Bonsoâââr, FO !, CHU, en tout cas, je dis -M-), Caroline (pour l'après, pour l'avant), Céline (personne n'est parfaite, sauf pour les bisous qui pètent), David (coreligionnaire de haut rang, chargé des relations avec la presse), Eric (plus connu sous le nom de EricMonPoteLeDentiste), Fabien ("– Le rock

c'est tout à fond !", "– OK, tiens, reprends un diabolo banane..."), Guillaume (Guitar Heroe), Julie (Wonder Julaïe, vaccin anti-groumou), Sylvaine (Si t'es pas contente, tu prends tes affaires et tu te casses... Mais après, tu reviens). Merci à vous, pour tout.

Finalement, je remercie mes parents, qui ont commencé à bricoler cette histoire dans leur coin, avec les moyens du bord, il y a déjà quelques années, et mes soeurs, parce que ce sont mes soeurs et qu'elles le valent bien.

The End.

Table des matières

Remerciements **1**

Table des matières **3**

Table des figures **7**

1 Introduction **9**
1.1 Détection d'intrusions 9
1.1.1 Sources de données 10
1.1.2 Méthodes d'analyse 12
1.1.3 Gestion des alertes 14
1.2 Problématique 15
1.3 Source de données 16
1.4 Contributions 18
1.5 Organisation de la thèse 21

2 État de l'art **23**
2.1 Détection d'intrusions vers les serveurs web 23
2.1.1 WebSTAT 24
2.1.2 Détection d'intrusions comportementale suivant les valeurs des paramètres 25
2.1.3 Visualisation de l'activité du serveur web 27
2.1.4 Diversification fonctionnelle 28
2.1.5 Détection d'intrusions par scénarios modulaire 30
2.1.6 Interaction entre un serveur web et une base de données 31
2.2 Combinaisons de méthodes de détection d'intrusions 32
2.3 Conclusion 34

3 Combinaison de méthodes de détection d'intrusions **35**
3.1 Formalisation du diagnostic des méthodes de détection d'intrusions comportementale et par scénarios 36
3.2 Étude des conflits de diagnostics 39
3.3 La combinaison $A_u \rightarrow M$ 41
3.4 La combinaison $M_i \rightarrow A$ 44

3.5 Conclusion 48

4 Le module de détection d'intrusions comportementale (*WebFilter*) 51
4.1 Source de données pour la détection 51
4.2 Le modèle de comportement 53
4.2.1 Méthodologie de construction du modèle comportemental 54
4.3 Partitionnement et analyse des groupes 64
4.3.1 Partitionnement des données issus du serveur web de Supélec . . 65
4.3.2 Étude des valeurs des variables et de leur variances 67
4.3.3 Étude des groupes 71
4.3.4 Profils liés aux groupes 73
4.3.5 Interprétation et choix des groupes pour l'intégration au modèle de détection comportementale 74
4.4 Conclusion 78

5 Le module de détection d'intrusions par scénarios (*WebAnalyzer*) 81
5.1 Précision et qualité du diagnostic 81
5.2 Principe de détection 83
5.2.1 Découpage et normalisation 84
5.2.2 Extraction de caractéristiques 85
5.2.3 Réconciliation et synthèse 86
5.3 Langage de signatures 87
5.3.1 Structure de signature 87
5.3.2 Organisation des signatures et procédé d'évaluation 90
5.4 Exemples de fonctionnement du WebAnalyzer 93
5.4.1 Tentative d'accès 94
5.4.2 Passage d'argument illicite 95
5.4.3 Exploitation de la vulnérabilité 95
5.5 Conclusion 96

6 Résultats expérimentaux 99
6.1 Le modèle de comportement de *WebFilter* 99
6.1.1 Le serveur web de l'IRISA 100
6.1.2 Un serveur web de France Télécom 113
6.2 Résultats d'analyse de *WebAnalyzer* 120
6.2.1 Le serveur web de Supélec 120
6.2.2 Le serveur web de l'IRISA 121
6.2.3 Un serveur web de France Télécom 122
6.3 Résultats obtenus avec la combinaison en série $A_u \rightarrow M$ 124
6.3.1 Le serveur web de Supélec 125
6.3.2 Le serveur web de l'IRISA 127
6.3.3 Un serveur web de France Télécom 129
6.4 Conclusion 131

7 Conclusion **133**
7.1 Contributions 133
7.2 Perspectives 135

A Données complètes **137**
A.1 Supélec 137
A.2 IRISA 139
A.3 France Télécom 145

Table des figures

1.1 Composition d'un système de détection d'intrusions selon l'IDWG . . . 10
1.2 Exemple de ligne de log HTTP 18

3.1 Diagnostic des méthodes de détection d'intrusions comportementale et par scénarios 36
3.2 Formalisation du diagnostic des méthodes de détection d'intrusions en fonction de la nature des événements analysés 37
3.3 Combinaison de méthodes de détection d'intrusions comportementale et par scénarios – les ensembles grisés sont des cas de figures impossibles car $E_s \cap E_i = \emptyset$; $X >> Y$ indique que le diagnostic émis par X est correct alors que celui émis par Y ne l'est pas ; $X > Y$ indique que le diagnostic émis par X est plus pertinent que celui émis par Y 39
3.4 Combinaison en série $A_u \rightarrow M$: utilisation en série d'une méthode de détection d'intrusions comportementale suivie d'une méthode de détection d'intrusions par scénarios. Les événements non qualifiés par la première méthodes sont analysés par la seconde. 41
3.5 Diagnostic de la combinaison en série $A_u \rightarrow M$ 43
3.6 Diagnostic final de la combinaison en série $A_u \rightarrow M$ 44
3.7 Combinaison en série $M_i \rightarrow A$: utilisation en série d'une méthode de détection d'intrusions par scénarios suivie d'une méthode de détection d'intrusions comportementale. Les événements reconnus comme intrusifs par la première méthode sont analysés par la seconde. 45
3.8 Diagnostic de la combinaison en série $M_i \rightarrow A$ 46
3.9 Diagnostic final de la combinaison en série $M_i \rightarrow A$ 47

4.1 Représentation des ressources sous forme de *trie* 53
4.2 Exemple d'histogramme des courbes de niveaux ou dendrogramme . . . 56
4.3 Exemple de partitionnements possibles en utilisant la méthode d'agrégation autour des centres mobiles 57
4.4 Observation des requêtes vers la ressource `/action.php` du serveur web de Supélec 61
4.5 Histogramme des indices de niveaux pour une CAH réalisées sur un échantillon de 25% des ressources du serveur web de Supélec 65

5.1 Signature Snort dédiée à la surveillance des accès au script `count.cgi` . 82
5.2 Trois requêtes HTTP vers le script `count.cgi` 83
5.3 Architecture des différents modules. 84
5.4 Exemple de signature utilisée par *WebAnalyzer* 87
5.5 Signature de recherche de shells 89
5.6 Réconciliation entre fichier et code retour 89
5.7 Exemple de sections permettant une détection conditionnelle 90
5.8 TITRE 91
5.9 Exemple de résultats d'analyse de *WebAnalyzer* relatifs à des tentatives d'accès au script `phf` 95
5.10 Exemple de résultats d'analyse de *WebAnalyzer* relatifs au passage d'argument illicite au script `phf` 96
5.11 Exemple de résultats d'analyse de *WebAnalyzer* relatifs à l'exploitation de la vulnérabilité du script `phf` 97

6.1 Histogramme des indices de niveaux pour une CAH réalisées sur un échantillon de 25% du premier sous-ensemble de données du serveur web de l'IRISA 101
6.2 Histogramme des indices de niveaux pour une CAH réalisées sur un échantillon de 25% du deuxième sous-ensemble de données du serveur web de l'IRISA 101
6.3 Histogramme des indices de niveaux pour une CAH réalisées sur un échantillon de 25% du troisième sous-ensemble de données du serveur web de l'IRISA 102
6.4 Histogramme des indices de niveaux pour la CAH réalisées sur les données du serveur web de France Télécom 114

Chapitre 1

Introduction

Les réseaux et applications informatiques sont utilisés de façon croissante et touchent des domaines aussi variés que les applications d'entreprise, le domaine médical ou le commerce électronique. Dans ce contexte, la sécurisation des systèmes d'information est une problématique de premier plan. La sécurité informatique a pour but d'assurer la confidentialité et l'intégrité des données ainsi que la disponibilité des services.

Pour assurer la sécurité des systèmes d'information, il existe toute une gamme de solutions telles que la cryptographie, l'application de politiques de sécurité utilisant des *firewall*, l'utilisation de serveurs mandataires. Cependant, ces mécanismes ne peuvent assurer complètement la sécurité des systèmes d'information. En effet, ils peuvent présenter des failles de conception, être mal configurés ou mettre en œuvre une politique de sécurité mal adaptée, ce qui permet à un attaquant de contourner les mesures de sécurité. Il est donc nécessaire d'utiliser en complément des outils permettant de détecter les attaques contre les systèmes d'information.

1.1 Détection d'intrusions

La détection d'intrusions consiste à révéler l'activité intrusive d'un attaquant vers, ou depuis, un système d'information grâce à l'observation de l'activité générée par les utilisateurs et les différents services. Les premiers travaux en détection d'intrusions ont débuté avec Anderson [4] en 1980 et Denning [15] en 1987. Une normalisation des systèmes de détection d'intrusions a depuis été proposée par l'*Intrusion Detection Working Group* (IDWG) [88].

La Figure 1.1 illustre les différents modules composant un système de détection d'intrusions, ou *intrusion detection system* (IDS). L'activité du système d'information est d'abord récupérée via des capteurs (*Sensor*), puis est transmise sous forme d'événements à un analyseur (*Analyzer*). Le module d'analyse est chargé de découvrir des traces d'intrusions dans l'activité du système d'information. Le cas échéant, il transmet une alerte à un gestionnaire d'alertes (*Manager*). Ce dernier est chargé de traiter les alertes émanant des différents analyseurs et de notifier toute activité suspecte sur le système d'information à l'opérateur de sécurité. Chacun des modules présenté est

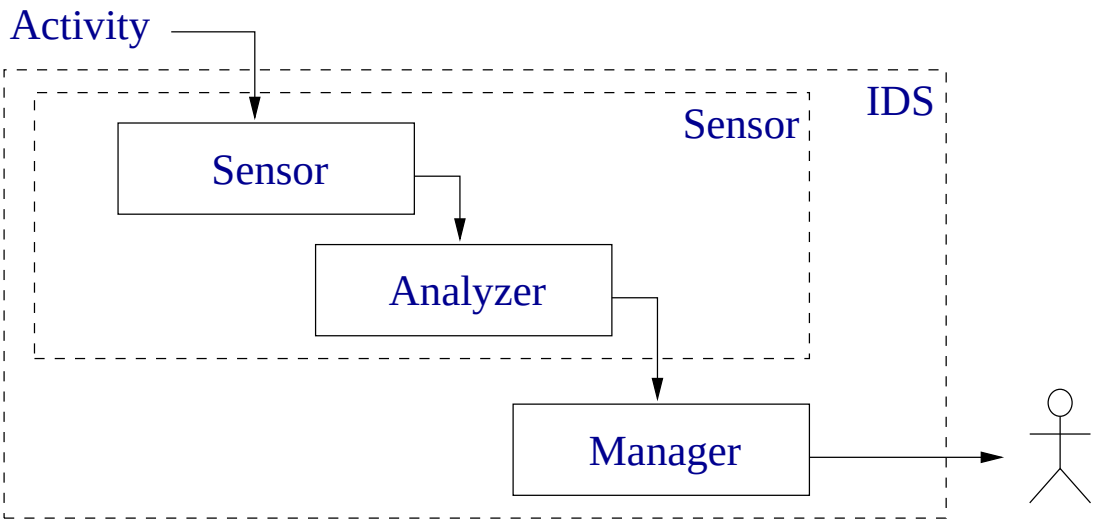

FIG. 1.1 – Composition d'un système de détection d'intrusions selon l'IDWG

configurable par un administrateur chargé d'appliquer la politique de sécurité destinée à la gestion des systèmes de détection d'intrusions.

Dans la suite de cette section, nous décrivons tout d'abord les différentes sources de données utilisées pour surveiller l'activité d'un système d'information. Nous évoquons ensuite les méthodes d'analyse utilisées pour mettre à jour des comportements intrusifs. Finalement nous abordons la gestion des alertes réalisée au niveau du gestionnaire (*Manager*).

1.1.1 Sources de données

La détection d'intrusions consiste à analyser une ou plusieurs sources de données afin d'y découvrir des manifestations d'attaques. Ces sources de données proviennent de l'activité des utilisateurs et de leurs interactions avec les services du système d'information. Le choix de la source de données peut être déterminant selon l'activité que l'on veut surveiller et le type d'attaque que l'on veut découvrir [9]. Nous détaillons les trois catégories de sources de données utilisées en détection d'intrusions : le trafic réseau, les audits systèmes et les audits applicatifs.

1.1.1.1 Trafic réseau

Les outils de détection d'intrusions réseau ou *Network-based Intrusion Detection System* (NIDS) écoutent de façon passive les données qui transitent sur le réseau. Ils permettent de découvrir des attaques qui se manifestent par une interaction sur le réseau entre un attaquant et une victime.

L'utilisation du trafic réseau pour la détection d'intrusions a plusieurs avantages. Ainsi, l'écoute passive du trafic n'engendre aucune interaction ou effet de bord sur

le trafic surveillé, les communications ne sont pas dénaturées et les performances du réseau ne sont pas altérées. De plus, en positionnant un outil de détection d'intrusions réseau sur un nœud de communication adéquat, on peut surveiller les communications destinées à un ensemble de services et de machines. Il est donc possible de surveiller toute l'activité d'un système d'information avec seulement quelques sondes de détection d'intrusions correctement disposées.

Un tel dispositif présente cependant des inconvénients. Tout d'abord, il ne permet de détecter que les attaques opérant via le réseau. Les attaques locales telles que le gain de droits administrateur par un simple utilisateur ne sont pas visibles. L'écoute réseau suppose aussi que les connexions ne soient pas chiffrées. En outre, il faut que l'outil de détection d'intrusions soit capable d'écouter tout le trafic qui transite sur son point d'écoute et d'analyser des connexions dont certains paquets peuvent être manquants [47]. Avec l'avènement des réseaux hauts et très hauts débits, le problème de la capacité d'écoute des outils de détection d'intrusions réseau se pose [16]. Les attaquants exploitent désormais les faiblesses liées à l'écoute du réseau et à certaines caractéristiques des protocoles réseaux [26, 68]. Une attaque peut se retrouver scindée en plusieurs fragments sur le réseau, les fragments prenant éventuellement des chemins différents sur le réseau. La sonde, si elle ne réassemble pas tous les fragments, peut ne pas reconnaître l'attaque. Si la victime reçoit l'ensemble des données, l'attaque peut réussir sans être détectée.

1.1.1.2 Audits système

Les outils de détection d'intrusions système (*HIDS : Host-based Intrusion Detection System*) analysent les données d'audit du système produites par une machine déterminée. Ces données d'audit retracent l'activité du système d'exploitation et des applications installées sur la machine [54]. Parmi ces données d'audit, on trouve par exemple les données *syslog* [85] et BSM [72]. D'autres outils utilisent, eux, directement les appels systèmes produits par l'exécution d'une application [86].

L'avantage de cette source de données est son exhaustivité, puisqu'elle retrace l'entière activité du système surveillé. De plus, la précision des données permet de retracer l'activité d'un attaquant.

Mais l'exhaustivité et la précision se paient en contrepartie par de grands volumes de données à traiter par un outil de détection d'intrusions. De plus, les ressources de la machine sont partagées entre son activité propre et la surveillance de cette activité. L'analyse des audits système peut avoir des répercutions non négligeables sur les performances du système surveillé.

1.1.1.3 Audits applicatifs

Les audits applicatifs sont directement fournis par les applications installées sur un système. Elles enregistrent tout ou partie des interactions avec les utilisateurs. Par exemple, les serveurs web enregistrent diverses informations sur les requêtes HTTP effectuées par les clients ainsi que le code de retour renvoyé par le serveur.

Cette source de données offre une information de haut niveau, synthétique et sémantiquement riche. Contrairement à l'écoute réseau ou à l'utilisation des audits système, la necessité de réassembler des ensembles d'événements pour identifier l'action d'un utilisateur est moindre. La consommation de ressources pour l'analyse en est d'autant réduite.

L'utilisation de cette source de données contraint cependant à effectuer une analyse dédiée à chaque application si on veut surveiller l'ensemble du système. Dans le cas contraire, un attaquant peut utiliser une interaction entre deux applications afin d'effectuer une attaque. De plus, il est nécessaire que les applications fournissent des données d'audit pertinentes pour la détection d'intrusions, ce qui n'est pas forcément le cas. Par exemple, si seul le service HTTP est surveillé, un attaquant peut remplacer un script présent sur le serveur web via une attaque contre le service FTP. Cette première étape de l'attaque n'est pas détectée, le service FTP n'étant pas surveillé. L'attaquant termine ensuite son attaque en exécutant le script malicieux avec une requête HTTP vers le serveur web. Cette seconde étape de l'attaque n'est pas détectée non plus puisqu'elle vise un script initialement présent sur le serveur web, mais dont le comportement a été modifié.

1.1.2 Méthodes d'analyse

La détection d'intrusions utilise deux familles de méthodes d'analyse pour surveiller les systèmes d'information. L'une, la détection d'intrusions par scénarios, est chargée de reconnaître les manifestations d'attaques connues, l'autre, la détection d'intrusions comportementale, vérifie la conformité des actions des utilisateurs par rapport à une norme précédemment définie.

1.1.2.1 Détection d'intrusions par scénarios

La détection d'intrusions par scénarios (ou *misuse detection*) consiste à reconnaître des manifestations d'attaques parmi les événements surveillés sur le système d'information. Pour ce faire, cette méthode d'analyse utilise une base de connaissances contenant des traces ou manifestations de scénarios d'attaques connues. Ces scénarios sont usuellement appelés *signatures d'attaques* et sont confrontés à l'activité du système d'information pour révéler la présence d'activité intrusive [28, 50].

Cette approche est basée sur la connaissance de scénarios d'attaques préalablement connues et permet d'obtenir un diagnostic précis. Une fois les manifestations d'attaques découvertes, la base de connaissances permet d'identifier l'attaque qui est mise en œuvre et l'opérateur peut ainsi mettre en place les contre-mesures nécessaires. La détection d'intrusions par scénarios ne permet de qualifier que des événements ou séquences d'événements intrusifs. La politique de sécurité implicite est que toute action qui n'est pas explicitement interdite est autorisée.

Cependant, cette approche, puisque basée sur la reconnaissance d'attaques connues, ne permet pas de découvrir de nouvelles attaques. Le maintien à jour de la base de connaissances et le travail de veille pour découvrir les nouvelles attaques sont donc

primordiaux.

Les systèmes experts peuvent être utilisés pour modéliser les attaques. Une base de règles traduit l'expertise d'un opérateur de sécurité. Une activité intrusive déclenche ces règles et engendre l'émission d'une alerte. On retrouve ce mécanisme dans P-BEST [50] (*Production-Based Expert System Toolset*) ainsi que dans le langage RUSSEL utilisé pour décrire les attaques dans le système ASAX [25].

Les attaques peuvent aussi être modélisées avec des automates et des machines à états finis. Les états correspondent à des états du système d'information et les transitions aux actions effectuées. Les automates décrivent l'enchaînement d'actions nécessaires à la réalisation d'une attaque [66]. Pouzol et *al.* [67] proposent un langage de scénarios d'attaques basé sur le formalisme de *parsing schemata*. La reconnaissance de scénarios d'attaques se fait ensuite via un système de ré-écriture. Cette technique permet de mettre en place des stratégies d'élagage basées sur des classes d'équivalence entre plusieurs occurrence d'un même scénario d'attaque.

La recherche de scénarios d'attaques se fait en identifiant un ou plusieurs événements. Cependant, les événements constituant l'activité surveillée ne sont pas statiques. Souvent, seule une caractéristique composant l'événement est nécessaire à l'identification de celui-ci comme faisant partie d'une activité intrusive. Par exemple, le fait qu'un utilisateur accède au fichier des mots de passe peut déclencher un événement critique, contrairement au cas où il accède à un de ses propres fichiers. Afin d'identifier précisément ces événements, la détection d'intrusions par scénarios est amenée à explorer le contenu des événements surveillés. La reconnaissance d'un événement intrusif se fait en utilisant la recherche de motifs [77] caractéristiques de l'exploitation d'une vulnérabilité. Le nombre de motifs à identifier croissant de concert avec le nombre d'attaques modélisées, la détection d'intrusions par scénarios tend à rechercher plusieurs motifs simultanément [44, 1].

1.1.2.2 Détection d'intrusions comportementale

La détection d'intrusions comportementale (ou *anomaly detection*) a pour charge de vérifier que les actions effectuées sur le système d'information sont conformes à un modèle de comportement sain préalablement établi. Une action, ou un ensemble d'actions, est declaré conforme s'il ne dévie pas de façon significative du modèle de comportement de référence. Dans le cas contraire, une alerte est émise. La construction du modèle de comportement peut se faire en étudiant par exemple le comportement usuel des utilisateurs avec des méthodes statistiques [27, 32] ou en établissant des règles [40, 84]. L'avantage de cette approche est qu'elle permet de découvrir toute forme d'attaques, y compris les nouvelles attaques.

Au contraire de la détection d'intrusions par scénarios, l'approche comportementale n'apporte que peu d'informations concernant l'activité intrusive, celle-ci n'étant pas qualifiée. Dans le cas où une alerte est émise, l'opérateur n'a pas d'informations concernant l'attaque subie ni les dommages qu'elle a pu causer. Il doit étudier les événements mis en cause afin de déterminer la vulnérabilité exploitée et les contremesures à mettre en œuvre. De plus, la constitution du modèle de référence et son

maintien à jour nécessitent un travail spécifique lié au type d'activité surveillée, au système d'information et à l'évolution du comportement des utilisateurs. Contrairement à la détection d'intrusions par scénarios, l'approche comportementale nécessite donc une configuration très spécifique et une adaptation constante à l'environnement surveillé. Les contraintes liées à la construction et au maintien à jour du modèle de comportement peuvent amener l'approche comportementale à émettre de nombreuses fausses alertes [49].

Denning [15] utilise les statistiques pour établir un profil d'utilisation de ressources pour un sujet donné. Si l'utilisation des ressources ne correspond pas aux observations statistiques de la période d'apprentissage, une alerte est émise.

On rencontre aussi des méthodes d'apprentissage automatiques où le modèle de comportement est constitué de règles de fonctionnement normal. Ces règles qui peuvent se présenter sous forme de séquences d'événements autorisés ou d'événements probables peuvent être construites en utilisant une approche d'apprentissage inductif [73] ou des réseaux de neurones [11].

Des approches immunologiques ont été proposées. Par exemple, Wespi et *al.* [87] modélisent l'état normal du système en utilisant des séquences d'événements de taille variable. Ces séquences d'événements sont générées grâce à l'algorithme Teiresias [69] issu de la bio-informatique.

Les techniques de fouille de données sont utilisées pour construire un modèle de comportement à partir de données d'apprentissage brutes. Le principe est de révéler des liens entre les événements afin de dégager des séquences d'événements sains ou des associations entre les applications et les accès aux données [48].

Les approches par spécifications n'utilisent pas de corpus d'apprentissage pour construire le modèle de comportement. Le comportement normal est spécifié grâce à un ensemble de règles établies, prenant pour modèle le mode de fonctionnement usuel de l'application surveillée ou la politique de sécurité [64].

1.1.3 Gestion des alertes

Pour le gestionnaire d'alertes, il s'agit tout d'abord de concentrer les alertes issues des différentes sondes de détection d'intrusions disposées sur le système d'information. Il doit ensuite traiter les alertes afin de présenter une information de haut niveau, intelligible pour un opérateur de sécurité, afin que celui-ci puisse prendre les contremesures nécessaires.

Ce module est nécessaire car les sondes de détection d'intrusions peuvent produire chacune une grande quantité d'alertes. Cet excès d'alertes peut provenir du caractère automatisé de certaines attaques. Par exemple, une activité de type *ver* générera un flot de plusieurs alertes pour chaque manifestation d'un ver, alors que ces alertes se réfèrent à une seule et même activité intrusive. La multiplication de l'activité du ver, due à son comportement répétitif, conduit à autant d'alertes émises. Un opérateur de sécurité peut se retrouver rapidement débordé par un trop grand volume d'alertes. A terme, l'opérateur de sécurité ne va examiner les alertes qu'épisodiquement et ne pourra plus appréhender le danger couru par le système d'information. Pour répondre à ce genre de

problème, Morin et *al.* proposent une méthode d'agrégation des séquences d'alertes [61] basée sur des chroniques [17]. Lorsqu'une séquence d'alertes valide une chronique, une alerte de plus haut niveau est émise, correspondant précisément à l'activité intrusive constatée.

Les sondes de détection d'intrusions, quelle que soit la méthode d'analyse utilisée, produisent un grand nombre de faux positifs. Ce problème est lié au souci d'exhaustivité des sondes de détection d'intrusions. Il vaut mieux, en effet, risquer d'émettre des fausses alertes plutôt que de manquer une attaque. Cependant, certains articles [35, 51, 18] évoquent des taux de fausses alertes allant de 90% à 99%. Dans ce contexte, il est possible qu'une ou plusieurs fausses alertes puissent participer à la validation d'une chronique. Dans ce cas, l'alerte issue de la chronique est elle-même une fausse alerte. Si, en plus du problème du trop grand nombre d'alertes, s'ajoute celui de la véracité de celles-ci, l'opérateur aura tôt fait de négliger les alertes émises par les sondes de détection d'intrusions.

Le traitement des volumes d'alertes est un problème de recherche en détection d'intrusions qui a été traité suivant de nombreuses approches. Manganaris et *al.* [58] utilisent le *data mining* pour caractériser le flot normal d'alertes provenant d'une sonde et établir un modèle de comportement. Ce dernier permet de détecter d'éventuelles anomalies dans le flot d'alertes en tenant compte du contexte dans lequel elles apparaissent. Il est ainsi possible de filtrer des alertes non pertinentes. Valdes et *al.* [78] proposent une approche par corrélation probabiliste, combinant des alertes en fonction de niveaux de similarité.

1.2 Problématique

Comme nous l'avons évoqué dans la Section 1.1, les problèmes en détection d'intrusions sont nombreux et la recherche dans ce domaine est très dynamique. Le problème de la qualité du diagnostic et de l'usage qui va pouvoir être fait des alertes [12] se pose de façon globale. Quelle source de données choisir pour surveiller le système d'information de façon optimale ? Quelle méthode de détection est la plus efficace en fonction des types d'événements surveillés ? Comment traiter efficacement toutes ces alertes ? Selon nous, la qualité du diagnostic en détection d'intrusions repose sur plusieurs critères :

1. l'analyse de l'ensemble de l'activité du service surveillé ;
2. la détection de l'ensemble des manifestations d'attaques ;
3. la génération minimale de fausses alertes.

Pour pouvoir effectuer une détection effectuer, il faut pouvoir collecter et analyser le plus d'activité possible d'un service. Tout événement non analysé constitue une mise en danger du système d'information puisqu'étant une attaque potentielle.

La capacité de détecter toutes les attaques demande d'une part d'étendre au maximum la couverture des signatures d'attaques utilisées, et donc leur capacité de détection, afin d'éviter toute tentative de dissimulation des attaques. Il s'agit aussi, d'autre part, d'être capable de détecter des attaques inconnues.

La capacité de ne générer que peu de fausses alertes et la capacité de détecter le plus de manifestations d'attaques peuvent sembler antagonistes. En effet, l'écriture de signatures d'attaques plus génériques, destinées à détecter l'exploitation d'une vulnérabilité dans différents contextes, conduit à émettre des alertes relatives à une activité saines, donc de fausses alertes.

Les travaux réalisés actuellement au niveau du gestionnaire d'alertes, en plus de chercher à réduire les volumes d'alertes, enrichir et améliorer l'information présentée à l'opérateur de sécurité, visent à éliminer les fausses alertes. Nous considérons cependant que le problème des fausses alertes doit être traité à la source et non pas relégué au niveau du gestionnaire d'alertes.

1.3 Source de données

Les serveurs web sont un vecteur d'échange d'information de plus en plus utilisé. Ce service, initialement destiné à délivrer des contenus statiques (pages web, image) à des clients, s'est enrichi de nombreuses fonctionnalités. Le protocole HTTP est devenu un support d'échange d'information, non seulement pour les applications web courantes (servir du contenu à un client), mais aussi pour effectuer des transactions commerciales (B2B, B2C). Les serveurs web mettent à disposition des clients des contenus dynamiques générés à la volée afin de proposer des contenus diversifiés et personnalisés. Le contenu dynamique est géré par des applications exécutées et hébergées sur le serveur. En plus des vulnérabilités propres au serveur web, l'exécution d'applications sur le serveur, ordonnée par un client distant, est un point d'entrée potentiel pour un attaquant, via, par exemple, l'utilisation malicieuse des arguments. Les serveurs web s'interfacent aussi avec des bases de données afin de stocker des données qui peuvent servir à la génération de contenus ou à des transactions. La présence de données sensibles, telles que des dossiers médicaux ou des coordonnées bancaires, constitue une motivation d'autant plus grande pour un attaquant.

Dans ce contexte, les serveurs web, les applications qu'ils hébergent et les données stockées en base sont des cibles à part entière et le serveur n'est plus seulement un point d'entrée vers le système d'information. Les différents serveurs web (par exemple Apache, IIS, Netscape) apportent chacun leur lot de vulnérabilités. De plus, le nombre croissant de failles de sécurité liées aux applications hébergés, rendent les serveurs web particulièrement attractifs comme cibles d'attaques. L'étude réalisée dans [76] montre que les vulnérabilités exploitées liées à l'activité web représentaient 23% du total des vulnérabilités sur la période d'avril 2001 à mars 2002. De plus, l'explosion du nombre de serveurs web[1] accroît considérablement le nombre de cibles pour un attaquant.

Les serveurs web étant par définition ouvert sur Internet, ils doivent être accessibles pour n'importe quel client. Dans ces conditions, les outils de sécurité préventifs usuels tels que les *firewall* et les *reverse-proxy* n'ont qu'un effet limité sur leur protection et les laissent exposés à des attaques directes. Dans ce contexte, la détection d'intrusions permet d'identifier ces attaques et de mettre en œuvre les contre-mesures nécessaires.

[1] `http://news.netcraft.com/archives/2005/06/01/june_2005_web_server_survey.html`

Afin de détecter les attaques contre les serveurs web, les outils de détection d'intrusions doivent surveiller les interactions entre le serveur web et les clients qui utilisent ce service. La source de données la plus souvent utilisée pour mettre en œuvre la détection d'intrusions est l'écoute du réseau. L'écoute des transactions HTTP via le réseau permet à l'IDS d'avoir accès à l'intégralité de l'échange entre le serveur web et le client. L'IDS dispose de toutes les informations nécessaires pour déterminer si une attaque a eu lieu, le succès de cette attaque et les informations qui sont renvoyées par le serveur web. En pratique, les IDS tel que Snort [70] n'utilisent pas la totalité de la communication pour déterminer si une attaque a lieu. La surveillance d'un serveur web via le réseau n'occasionne aucune interférence avec le service. L'IDS écoute le réseau de façon passive et autonome, il n'interfère pas dans les capacités de traitement de la requête du client puisqu'il ne partage pas de ressource avec le serveur web. Cependant, si la transaction entre le client et le serveur web est chiffrée, l'IDS n'a pas accès aux données transitant entre le client et le serveur web et la détection d'intrusions n'est plus possible. De plus l'IDS peut être trompé par un attaquant utilisant des techniques dites d'évasion telles que la segmentation TCP par exemple [71]. Si le trafic ne peut être analysé par l'IDS, ce dernier est inefficace.

Outre l'écoute du réseau, les autres sources de données pour assurer la sécurité des serveurs web se trouvent sur la machine qui héberge le serveur. Il est ainsi possible d'intégrer un IDS directement au sein du serveur web, en l'intégrant à la chaîne de traitement de celui-ci. Il accède ainsi à l'intégralité de la transaction HTTP entre le client et le serveur web. Le contenu de la transaction est déchiffré au niveau du serveur et ré-assemblé pour être traité au niveau applicatif par le serveur web. Positionner un IDS dans le serveur web nous affranchit des problèmes rencontrés avec l'écoute du réseau. De plus, cela rend l'IDS capable d'arrêter une attaque en cours puisqu'il se trouve dans la chaîne de traitement de la transaction avec le client. Cependant cette position risque d'augmenter de façon trop significative le temps de réponse du serveur au client. L'intégration à un serveur web nécessite un effort de développement propre à chaque type de serveur, voir même pour chaque version de serveur.

Il est possible de surveiller l'activité du serveur web en utilisant les fichiers de log où il enregistre son activité. L'IDS se positionne alors comme un consommateur des fichiers de log pour analyser l'activité du serveur web et de ses clients. L'utilisation des fichiers de log fournit une information synthétique de haut niveau qui décrit l'action effectuée par le client et la réponse faite par le serveur. L'IDS peut se positionner comme consommateur direct des fichiers de log, auquel cas l'analyse se fait en ligne. Il est aussi possible d'analyser ces fichiers hors ligne pour effectuer une analyse *post-mortem* d'un serveur web compromis par exemple. Finalement l'IDS peut effectuer une analyse « séquentielle » du fichier de log afin de n'utiliser les ressources partagées avec le serveur que de façon ponctuelle. L'utilisation des fichiers de log contraint l'IDS à analyser une requête qui a déjà été traitée par le serveur web. Il n'est pas possible d'empêcher une attaque. Les fichiers de log ne contiennent qu'une partie de la transaction entre le client et le serveur web, et l'IDS n'a donc pas accès à l'ensemble de la transaction web. De plus, l'utilisation du seul fichier de log ne permet pas à l'IDS d'avoir des informations concernant le serveur web, telles que le système d'exploitation, le type de serveur utilisé,

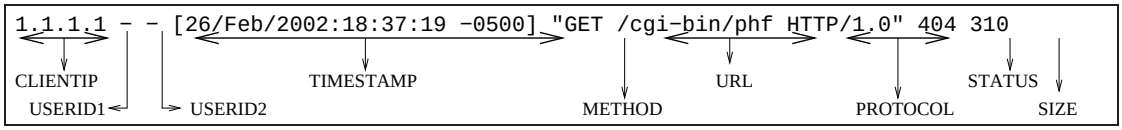

FIG. 1.2 – Exemple de ligne de log HTTP

les applications hébergées ou encore l'organisation des données. Finalement, l'utilisation de cette source de données assujettit à la présence de la ligne de log correspondant aux actions des clients. Si une attaque permet à un client de ne pas laisser de trace dans les fichiers de log, celui-ci ne peut pas être détecté. Cependant nous considérons qu'une attaque réussie contre un serveur web nécessite une réponse de sa part et donc qu'il enregistre cette action dans les fichiers de log.

Nous avons choisi d'utiliser les fichiers de log pour surveiller l'activité d'un serveur web. L'information fournie par les fichiers de log ne nécessite que peu de pré-traitement pour analyser les actions effectuées par les clients. Même si toute la transaction HTTP entre le client et le serveur web n'est pas présente, les fichiers de log contiennent la ressource visée par le client, donc la cible potentielle d'attaque et le code de retour émis par le serveur. Ces informations nous semblent suffisantes pour effectuer une analyse efficace.

Chacune des lignes d'un fichier de log représente une requête effectuée par un client vers le serveur web. Nous utilisons le format *Common Log Format* (CLF) [55] qui peut être généré par l'ensemble de serveurs web connus de l'auteur. La Figure 1.2 montre un exemple de ligne de log au format CLF. Le tableau 1.1 recense la signification des différents champs observés dans la Figure 1.2.

Une version étendue du format CLF, *Extended Common Log Format* (ECLF), contient des champs supplémentaires tels que le *referrer* qui indique la page précédemment visitée par le client, le type de navigateur et le système d'exploitation du client. Ces données sont fournies automatiquement par le navigateur du client et peuvent être altérées ou masquées par un attaquant. Nous ne les utilisons donc pas dans notre processus de détection d'intrusions.

1.4 Contributions

Nous étudions dans cette thèse l'opportunité de combiner des méthodes de détection d'intrusions comportementale et de détection d'intrusions par scénarios afin d'améliorer la qualité du diagnostic en détection d'intrusions. Notre première contribution relève du choix d'une combinaison. Pour ce faire, nous avons formalisé le diagnostic des deux méthodes de détection d'intrusions et étudié les capacités de diagnostic de chacune des combinaisons possibles. A partir de cette étude, de nos objectifs et de la composition des données que nous voulons traiter, nous pouvons déterminer quelle est la combinaison la plus efficace pour répondre à nos besoins. La combinaison choisie analyse en premier lieu les événements avec une méthode de détection d'intrusions comportementale. Les

Identifiant	**Signification**
CLIENTIP	Adresse apparente du navigateur d'où provient la requête. Cette adresse peut être en fait celle d'un serveur mandataire (proxy) ou d'un pare-feu. Ce champ est systématiquement renseigné.
USERID1	Identifiant utilisateur résultant d'une identification sur le domaine auquel appartiennent l'utilisateur et le serveur (identification sur un domaine NT par exemple). Ce champ est très rarement renseigné, et n'est utilisé que dans des réseaux privés d'entreprise [33].
USERID2	Identifiant utilisateur résultant d'une identification HTTP (présentation d'une demande de nom et de mot de passe à l'utilisateur). Ce champ est rarement renseigné.
TIMESTAMP	Horodatage de la requête par le serveur. Exprimé le plus souvent en heure locale du serveur, avec décalage GMT.
METHOD	Méthode utilisée pour la requête. La liste des valeurs acceptables est définie dans la spécification du protocole HTTP [63]. Cette liste de valeurs pourra être complétée pour d'autres protocoles (par exemple WebDAV [24]).
URL	Requête présentée par l'utilisateur.
PROTOCOL	Indique la version du protocole utilisée par le navigateur. Le protocole est en général une des trois valeurs HTTP/0.9, HTTP/1.0 ou HTTP/1.1.
STATUS	Indique la manière dont le serveur a traité la requête, à partir d'une liste spécifiant la signification des codes [63].
SIZE	Nombre d'octets renvoyés par le serveur au navigateur.

Tab. 1.1 – Description d'une ligne de log d'un serveur HTTP

événéments non reconnus comme sain par cette première méthode sont ensuite analysés par une méthode de détection d'intrusions par scénarios. Cette combinaison est notée $A_u \rightarrow M$.

La formalisation que nous proposons peut être utilisée pour déterminer la combinaison la plus adéquate pour d'autres objectifs et en fonction d'autres types de données à analyser. Nous avons choisi d'expérimenter la combinaison de méthodes de détection d'intrusions pour analyser l'activité des serveurs web. La formalisation, une justification du choix de notre combinaison et une première mise en œuvre de cette combinaison ont été publié dans [75].

La seconde contribution concerne la mise en œuvre d'un outil de détection d'intrusions comportementale, *WebFilter*, capable de reconnaître rapidement des événements sains. Dans le contexte de la combinaison d'outils de détection d'intrusions, *WebFilter* apporte des capacités de filtrage efficace. De plus, nous proposons une méthodologie de construction du modèle comportemental basée sur des algorithmes statistiques robustes et éprouvés. Cette méthodologie ne nécessite qu'une expertise limitée dans le domaine des serveurs web. Elle offre en plus un gain de temps substantiel comparée à une construction manuelle du modèle de comportement, tout en n'engendrant que très peu de faux négatifs. Une présentation partielle de *WebFilter* a été publiée dans **??**.

La troisième contribution est relative à un outil de détection d'intrusions par scénarios dédié à l'analyse de l'activité des serveurs web : *WebAnalyzer* [13, 14]. *WebAnalyzer* a pour particularité d'offrir un diagnostic très précis en utilisant non seulement des signatures révélant l'exploitation de vulnérabilités, mais aussi en contextualisant les attaques. Il est possible de déterminer l'objectif de l'attaque ainsi que sa portée. La précision du diagnostic est possible grâce à l'utilisation d'un grand nombre de signatures. Afin d'optimiser le traitement lié à l'analyse des événements, nous proposons un modèle d'organisation des signatures, ainsi qu'un mode de détection conditionnel. L'ensemble des informations apporté par *WebAnalyzer* lui permet finalement d'associer un niveau de sévérité à chaque attaque et donc de hiérarchiser les alertes présentées à l'opérateur de sécurité.

La quatrième contribution concerne l'implémentation de la combinaison $A_u \rightarrow M$ et les expérimentations réalisées sur trois serveurs web. Pour chacun des serveur web, nous présentons et étudions les résultats obtenus pour la création de leur modèles de comportement respectifs. Ces modèles de comportement sont destinés à être utilisés par *WebFilter* dans la combinaison $A_u \rightarrow M$. Nous présentons ensuite les résultats obtenus avec *WebAnalyzer* seul. Nous présentons ensuite les résultats obtenus avec la combinaison $A_u \rightarrow M$ et en utilisant les modèles de comportement préalablement établis. Les résultats montrent que l'utilisation de la combinaison $A_u \rightarrow M$ permet de répondre efficacement au problème des faux positifs, ainsi que d'améliorer la qualité du diagnostic.

1.5 Organisation de la thèse

Nous présentons dans le Chapitre 2 un état de l'art des travaux portant sur la détection d'intrusions dédiée aux serveurs web, ainsi que des travaux relatifs à la combinaison de méthodes de détection d'intrusions.

Dans le Chapitre 3, nous étudions de façon formelle les combinaisons possibles de méthodes de détection d'intrusions comportementale et de détection d'intrusions par scénarios. A partir de cette étude, nous montrons que la combinaison la plus appropriée est la suivante : les événements sont tout d'abord analysés par une méthode de détection d'intrusions comportementale ; les événements qui ne sont pas déclarés sains sont ensuite analysés par la méthode de détection d'intrusions par scénarios.

La mise en œuvre de l'outil de détection d'intrusions comportementale, *WebFilter*, est présentée dans le Chapitre 4. Ce chapitre décrit la capacité de notre outil à reconnaître rapidement et efficacement les événements sains, ainsi que la méthodologie mise en œuvre pour créer le modèle de comportement à partir d'un historique d'apprentissage opérationnel (c'est-à-dire pouvant contenir des attaques et des erreurs).

Le Chapitre 5 décrit l'outil de détection d'intrusions par scénarios, *WebAnalyzer*, utilisé dans notre combinaison. Cet outil répond aux besoins de diagnostic précis, à savoir une description exhaustive de l'attaque et de la vulnérabilité utilisée ainsi que la portée de l'attaque. De plus, il intègre des mécanismes permettant d'évaluer la sévérité d'une attaque.

Nous présentons dans le Chapitre 6 les résultats expérimentaux. Nous décrivons dans ce chapitre les résultats obtenus en utilisant l'outil de détection d'intrusions par scénarios seul et dans la combinaison.

Finalement, nous concluons les travaux effectués dans cette thèse avec le Chapitre 7 et nous présentons différentes perspectives pour étendre et améliorer les résultats obtenus.

Chapitre 2

État de l'art

Nous présentons dans ce chapitre des travaux dont les résultats sont en rapport direct avec les travaux de cette thèse.

Dans un premier temps, nous détaillons les travaux mettant en œuvre des méthodes de détection d'intrusions dédiées à l'analyse de l'activité des serveurs web. Dans un deuxième temps, nous étudions des travaux relatifs à la combinaison de méthodes de détection d'intrusions et à la collaboration d'outils de détection d'intrusions.

2.1 Détection d'intrusions vers les serveurs web

Les outils de détection d'intrusions ont tendance à se spécialiser. En effet, il semble plus facile de surveiller une activité précise, possédant un ensemble de caractéristiques bien déterminées. Cette spécialisation permet de mettre en œuvre une analyse plus efficace et de répondre à des problématiques bien précises. On note par exemple les travaux de Vigna et *al.* [81] visant les réseaux *ad hoc* utilisant le protocole de routage AODV ou les travaux de Zhang et *al.* relatifs au protocole BGP [89].

Les serveurs web, au-delà d'un simple service de présentation de contenu, sont devenus de véritables vecteurs d'information. Les applications web sont désormais des interfaces d'utilisation courantes. Pour le particulier, il est possible d'acheter des produits commerciaux en ligne et de se voir offrir des services personnalisés. Pour les entreprises, l'utilisation d'applications hébergées sur un serveur web permet un déploiement rapide et à moindre frais, ainsi qu'une centralisation de la gestion de l'information.

L'utilisation croissante des serveurs web en font des cibles de choix puisqu'ils concentrent beaucoup d'informations, qu'elles soient relatives à des personnes (coordonnées bancaires par exemple), ou a des entreprises (gestion des stocks ou publications internes par exemple). De plus, la diversité des applications hébergées constitue autant de cibles potentielles pour un attaquant. Il semble donc nécessaire de protéger ce service de façon spécifique.

Petite introduction au vocabulaire web

Dans la suite nous allons détailler des travaux relatifs à la sécurité des serveurs web et utiliser un vocabulaire propre à cette activité. Dans un souci de clarté, nous rappelons succintement le mode de fonctionnement d'un serveur web ainsi que le vocabulaire idoine. Des caractéristiques plus détaillées seront abordées dans les chapitres suivant, notamment au Chapitre 5.

La communication entre un serveur web et un client est toujours initiée par ce dernier. Le client émet une requête HTTP [63, 21] vers une ressource potentiellement hébergée sur le serveur web. La méthode utilisée (`GET` ou `POST` par exemple) indique le mode de communication utilisé entre le client et le serveur. La ressource est identifiée par un chemin dans l'arborescence du système de fichiers du serveur web. Une ressource peut être statique ou dynamique.

Une ressource statique correspond à des données que le serveur web récupère directement dans son système de fichier et qu'il renvoie telles quelles au client.

Une ressource dynamique correspond au résultat de l'exécution d'une application hébergée sur le serveur web. Lorsque qu'une application est visée par la requête d'un client, celle-ci est exécutée par et sur le serveur web afin de générer dynamiquement le contenu destiné au client. Le client peut passer des paramètres à l'application qui sont soit ajoutés à la suite du chemin de la ressource (`http://www.monserveur.com/monapplication?var1=val1&var2=val2`), soit inscrit dans le corps de la requête HTTP.

En plus du contenu, le serveur web renvoie un code de statut (*status code*) au client afin de lui préciser si sa requête a réussi ou pas. En cas d'échec, différentes valeurs de *status code* permettent au client de déterminer quel type d'erreur s'est produite sur le serveur (par exemple `404` si la ressource n'existe pas).

2.1.1 WebSTAT

Les premiers travaux que nous présentons sont ceux réalisés par Vigna et *al.* avec l'outil de détection d'intrusions par scénarios *WebSTAT* [82]. Cet outil présente plusieurs caractéristiques intéressantes. Tout d'abord, il utilise plusieurs sources de données liées à l'activité du serveur web. Comme nous l'avons décrit dans l'introduction, il est possible d'utiliser trois types de sources de données en détection d'intrusions : le trafic réseau, les audits système et les audits applicatifs, chacune de ces sources de données ayant ses avantages et ses inconvénients. Le choix fait ici est d'utiliser ces trois sources de données afin de capter le plus d'informations susceptibles de révéler la présence d'une attaque. Chaque événement destiné ou issu de l'activité du serveur web est utilisé dans la phase de détection. Les événements provenant des trois sources de données différentes sont tout d'abord normalisés afin d'être appréhendés par le processus d'analyse mis en œuvre dans *WebSTAT*. Ils sont ensuite mis dans une file d'attente avant d'être analysés.

WebSTAT s'appuie sur un langage de description d'attaques permettant de décrire des attaques complexes. Il est capable de prendre en compte l'état du serveur web au moment de l'analyse de son activité en s'appuyant sur l'analyse des états et transi-

tions (STAT - *State-Transition Analysis Technique*) présentée dans [28]. Le langage de description d'attaques STATL [19, 20] est développé suivant cette technique d'analyse. La mise en œuvre initiale était destinée à effectuer une analyse du trafic réseau et des audits système [80]. Pour les besoins de *WebSTAT*, l'implémentation a été enrichie d'un module permettant de prendre en compte les fichiers de log d'un serveur web.

Cette approche permet de décrire les différents états du serveur web en fonction de son activité. Les signatures d'attaques sont représentées sous forme d'automates décrivant les différentes étapes d'une attaque et les transitions nécessaires pour que l'attaque soit menée à son terme.

La multiplication des sources de données semble être une bonne méthode pour éviter les tentatives d'évasion, ou tout simplement de se prémunir du manque d'information lié à une source de données unique. Cependant, la diversification des sources de données amène l'outil à gérer des duplicatas d'information, et donc occasionne une charge de traitement supplémentaire pour l'outil.

L'utilisation du langage STAT permet de détecter des attaques se déroulant en plusieurs étapes. Cependant, cette approche ne semble appropriée que pour une faible proportion des attaques contre les serveurs web. En effet, le protocole HTTP est un protocole sans état. De plus, l'utilisation du langage STAT pour la description des attaques peut rendre l'écriture de nouvelles signatures difficile pour un opérateur de sécurité. Cette difficulté semble être implicitement prise en compte par les auteurs, puisque *WebSTAT* utilise aussi le *pattern matching* de façon mono-événementielle pour détecter des signatures d'attaques.

2.1.2 Détection d'intrusions comportementale suivant les valeurs des paramètres

Kruegel et *al.* [43, 42] s'intéressent à une méthode de détection d'intrusions comportementale pour les serveurs web. Elle se base sur les valeurs des paramètres envoyés aux applications hébergées par le serveur. Ces travaux sont une application spécifique aux serveurs web de travaux présentés dans [41] où Kruegel et *al.* cherchent à mettre à jour des attaques en analysant les arguments passés aux appels système.

La méthode décrite prend en entrée les fichiers de log générés par le serveur web. Elle associe une « valeur de normalité » (*anomaly score*) aux requêtes vers des applications hébergées sur le serveur. La méthode de détection proposée par Kruegel et *al.* ne vise que les paramètres passés aux applications hébergées sur le serveur web. Les requêtes vers des ressources statiques ne sont donc pas concernées. De même, les requêtes envoyées avec la méthode `POST` ne sont pas traitées, leurs paramètres n'apparaissant pas dans les fichiers de log. La phase de détection s'effectue de façon mono-événementielle. Chaque événement est une requête vers une application hébergée sur le serveur et une combinaison ordonnée de paramètres et de leurs valeurs associées.

La méthode de détection d'intrusions comportementale se base sur une conjonction de critères qu'une valeur doit remplir pour être déclarée saine ou non. Chacun de ces critères détermine une probabilité pour la valeur analysée. C'est la combinaison de ces probabilités qui détermine si une valeur de paramètre est acceptable ou pas en fonction

d'un seuil établi lors d'une phase d'apprentissage. La multiplication des critères a pour but de combiner leurs capacités de détection et de compenser leurs faiblesses respectives. Ces critères sont au nombre de six.

Le premier critère émet une probabilité relative à la longueur de la valeur associée à un paramètre. Les valeurs des paramètres sont souvent, d'après les auteurs, de petite taille. Ce critère permet de détecter, par exemple, les tentatives de *buffer overflow* qui contiennent une grande quantité de code binaire destiné à être exécuté indûment sur le serveur. La deuxième probabilité concerne la distribution des fréquences des caractères composant la valeur d'un paramètre. Ce critère permet de parer aux tentatives d'évasion visant la valeur d'un paramètre, comme par exemple l'encodage de code binaire. Le troisième critère émet une probabilité en rapport avec la structure de la valeur du paramètre. Ce critère applique une inférence structurelle sur les caractères composant la valeur en utilisant les modèles de Markov et la probabilité Bayésienne. La quatrième probabilité émise est relative à la présence de valeur énumérée. En effet, les valeurs possibles pour un paramètre sont souvent prédéfinies par les spécifications de l'application hébergée sur le serveur. L'utilisation d'une valeur non prédéfinie par l'application est symptomatique d'une activité intrusive. Le cinquième critère concerne la présence ou l'absence d'un paramètre pour l'utilisation d'une application. Les applications web construisent les requêtes qui vont leur être adressées, charge à l'utilisateur d'affecter des valeurs aux paramètres. Dans ces conditions, l'absence ou la présence d'un paramètre surnuméraire est un indice d'activité suspecte. Le sixième critère examine l'ordre des paramètres. Comme précédemment, les requêtes sont construites de façon automatisée et l'ordre des paramètres est *a priori* déterminé. Un changement d'ordre indique une manipulation des paramètres et donc une activité potentiellement suspecte.

Les expérimentations menées par les auteurs sur trois serveurs web (deux serveurs web académiques et un industriel, Google), montrent la capacité de leur méthode à détecter non seulement des attaques avérées contre ces serveurs web, mais aussi des utilisations non conventionnelles des applications hébergées. De plus, le taux de fausses alertes, inférieur à 10^{-3}, est extrêmement faible pour un outil de détection d'intrusions comportementale.

Cependant l'approche proposée par Kruegel et *al.* présente plusieurs défauts. Tout d'abord, comme toute méthode de détection d'intrusions comportementale, elle permet seulement de désigner des requêtes suspectes, et n'apporte aucune qualification concernant l'attaque potentiellement menée. L'opérateur de sécurité est chargé de déterminer le type d'activité intrusive et la portée de l'attaque. D'autre part, les données utilisées lors de la période d'apprentissage sont censées être exemptes de toute activité intrusive, ce qui suppose une analyse complète et approfondie de ces données. Aucune information n'est donnée concernant cette analyse. Les conséquences de la multiplication des critères ne sont pas discutées par les auteurs. Cette multiplication peut impliquer que la constitution du modèle de comportement soit coûteuse en terme de ressources. De même, pour la phase de détection, les auteurs ne discutent pas des conséquences possibles concernant les performances lors de l'analyse. Finalement, les auteurs ne discutent pas de la pérennité du modèle de comportement qu'ils établissent avec leur approche. Dans

ce cas, cette question peut être importante vu le nombre de critères d'apprentissage mis en œuvre.

2.1.3 Visualisation de l'activité du serveur web

Parmi les travaux de thèse d'Axelsson dédiés à la visualisation en détection d'intrusions [8], celui-ci propose d'utiliser la réduction de log et un outil de visualisation pour faire de la détection d'intrusions comportementale spécifique aux serveurs web [7].

Les fichiers de log utilisés contiennent les requêtes des clients au format CLF [55]. La réduction de log s'effectue en deux phases. La première phase consiste à éliminer les duplicatas, ne conservant ainsi qu'une seule instance de chaque requête. La seconde phase est basée sur des statistiques descriptives et plus particulièrement sur la fréquence d'apparition des événements. Les instances de requêtes sont, tout d'abord, découpées suivant les caractères suivants : " `? :\&=+\$/`". Ces caractères ont une signification réservée pour la construction d'une requête HTTP. Ils servent de séparateurs pour scinder la requête émise par un client. Le découpage permet de séparer la méthode, les répertoires composant le chemin vers la ressource, les paramètres et leur valeur et la version de protocole utilisée. Par exemple, la requête `GET /pub/index.html HTTP/1.1` est découpée en éléments comme suit : `GET`, `pub`, `index.html`, `HTTP` et `1.1`. L'auteur compte ensuite la fréquence d'apparition de ces éléments et détermine une valeur limite pour la sélection. Les éléments apparaissant très fréquemment sont des éléments d'utilisation courante dans les requêtes émises par le serveur web et donc des éléments probablement non représentatifs d'activité suspecte. De plus, les éléments ayant une grande fréquence d'apparition peuvent fausser les moyennes et masquer l'activité intrusive qui n'apparaît que plus rarement. Les instances de requêtes sont ensuite ordonnées suivant leur fréquence de demande croissante et seules les moins fréquentes sont sélectionnées suivant le seuil déterminé précédemment pour être visualisées (5 200 instances de requêtes sur 220 000 dans l'article [7]). L'hypothèse est que l'activité instrusive est moindre comparée à l'activité globale du serveur et que c'est en observant les instances de requêtes ayant une faible fréquence qu'on a le plus de chances de trouver des attaques.

La visualisation se fait grâce au découpage que nous avons vu précédemment. Les éléments composent les noeuds d'un graphe orienté. Suivant l'exemple précédent, l'élément `GET` est relié à l'élément `pub` qui est relié à l'élément `index.html`. Celui-ci est relié à l'élément `HTTP`. Ce dernier est finalement relié à l'élément `1.1`. Chaque instance de requête décrit alors un chemin au sein du graphe orienté, chaque noeud étant unique. La visualisation ainsi déterminée laisse apparaître des *patterns* ou structures visuelles regroupant des instances de requêtes ayant les mêmes caractéristiques. Le procédé de détection consiste à identifier visuellement des *patterns* et à déterminer la nature des instances de requêtes les composant. Ces instances ayant des caractéristiques communes, il est *a priori* possible pour l'opérateur de sécurité de classifier les instances de requêtes composant un même *pattern*. Dans l'expérience qu'il décrit, l'auteur détecte en premier lieu un *pattern* relatif à une tentative de détournement de ressource destiné à envoyer des mails non sollicités (*spam*).

En pratique, selon l'auteur, l'opérateur de sécurité détermine plus facilement quels sont les *patterns* d'instances de requêtes saines et les élimine de la visualisation. En effet, l'auteur observe que les *patterns* d'instances de requêtes sains ont une forme arborescente facilement identifiable, ce qui n'est pas le cas des attaques. Cette dernière caractéristique s'explique par le fait que les attaques partagent souvent des caractéristiques communes qui se visualisent seulement par des nœuds communs et non pas par des chemins communs dans le graphe. Après avoir éliminé les *patterns* d'instances de requêtes saines, il reste à l'opérateur de sécurité à analyser les instances de requêtes intrusives ou suspectes. La notion de *patterns* permet d'identifier des classes d'attaques dont est victime le serveur web.

Selon l'auteur, l'effort d'identification et d'élimination des *patterns* d'instances de requêtes saines n'est nécessaire qu'une seule fois. En effet, une fois les requêtes saines identifiées, elles n'apparaissent plus dans les futures visualisations. De plus, l'apparition de nouvelles ressources sur le serveur au fil du temps, et donc de nouvelles instances de requêtes, est relativement limité et donc plus facile à appréhender.

L'approche proposée par Axelsson n'est pas à proprement parler une méthode de détection d'intrusions. En effet, l'effort de détection est réalisé par un opérateur de sécurité. La qualité de la détection est directement liée à l'expertise de l'opérateur de sécurité. De plus, le procédé d'élimination des instances de requêtes saines repose sur une reconnaissance visuelle de *patterns*. Ce procédé ne garantit pas qu'un *pattern* donné ne contienne pas d'instances de requête intrusives, ni que l'opérateur ne se trompe dans son interprétation. En outre, cet outil, basé exclusivement sur la visualisation, ne propose pas de méthode de délimitation de *pattern*, c'est-à-dire que selon deux opérateurs différents, la visualisation des *patterns* peut être différente. Finalement, même si cette approche apporte une classification implicite des instances de requêtes intrusives, elle ne met en œuvre ni la qualification des instances de requêtes, ni la qualification des classes d'instances formées.

2.1.4 Diversification fonctionnelle

La programmation N-versions [6] consiste à faire développer plusieurs versions d'un même service par des équipes différentes. Elle permet de détecter et de tolérer des erreurs de fonctionnement en comparant les sorties des différentes versions développées. Les différences détectées sont dans ce contexte des erreurs de conception de l'une des versions du service développé.

Majorczyk et *al.* [57, 56] proposent d'utiliser cette méthode pour mettre en œuvre un mécanisme de détection d'intrusions spécifique aux serveurs web.

Contrairement aux principes de la programmation N-versions, les différentes versions de serveurs web utilisées par Majorczyk et *al.* ne sont pas des développements spécifiques mais des produits déjà existants sur le marché, ou *Components-Off-The-Shelf* (COTS). Selon les auteurs, les COTS étant des produits largement testés, les fautes de conception propre sont très rares. Les serveurs COTS utilisés sont choisis de sorte que l'intersection entre les ensembles de vulnérabilités (potentielles ou avérées) de chaque version soit nulle ou presque. Pour ce faire, les auteurs recommandent d'utiliser des serveurs à

la fois diversifiés au niveau des applications hébergées par les serveurs, des systèmes d'exploitation et au niveau matériel. Le fonctionnement des serveurs web étant basé sur le même protocole de communication (HTTP), la détection de l'activité intrusive se fait en analysant les différences entre les réponses émises par les serveurs web à une même requête. L'hypothèse de ces travaux est que les différences détectées entre les réponses émises par les serveurs web sont soit des différences d'interprétation des spécifications par les développeurs, soit des intrusions visant un serveur web spécifique. Considérons par exemple qu'un des serveurs web soit vulnérable à l'attaque du ver Nimda alors que les autres ne le sont pas. En cas d'attaque de ce vers, le serveur web vulnérable renvoie une réponse positive, alors que les autres serveurs renvoient un message d'erreur standard.

L'architecture proposée par Majorczyk et *al.* est la suivante : les requêtes des clients sont adressées à un proxy qui est le seul point d'entrée visible des clients. Chaque requête est ensuite transmise à l'ensemble des COTS qui renvoient leurs réponses au proxy où elles sont analysées avant qu'une réponse ne soit envoyée au client. L'analyse consiste à comparer un certain nombre de champs tels que le *status code*, différents en-têtes choisis et le corps de la réponse par exemple. Si une majorité de réponses identiques se dégage, la réponse correspondante est envoyée au client. Les réponses ne faisant pas partie de la majorité occasionnent chacune une alerte. Dans le cas où aucune majorité n'est trouvée, le proxy renvoie un message d'erreur standard au client.

Afin de faire face aux différences liées à l'utilisation de COTS en lieu et place de développements spécifiques, les auteurs utilisent des règles de masquage, évitant ainsi l'émission d'un trop grand nombre de fausses alertes. En entrée, les requêtes émises par les clients peuvent être modifiées pour les adapter à chaque COTS. En sortie, avant d'être analysées par l'IDS, les réponses émises par les COTS peuvent être modifiées afin de pallier les différences de fonctionnement connues entre les différentes implémentations de serveurs web. En pratique, les auteurs utilisent trente-six règles de masquage afin d'éviter de générer des fausses alertes.

L'utilisation de règles de masquage est censée répondre au problème des faux positifs en masquant certaines différences entre les réponses des différents serveurs web. Cependant les auteurs n'évaluent pas la portée des règles de masquage sur la génération de faux négatifs. En effet, la diversification fonctionnelle utilise les différences entre les réponses des COTS pour détecter une activité intrusive. Si ces différences sont masquées, la détection peut être infructueuse. D'autre part, les auteurs préconisent de diversifier les COTS à tous les niveaux, c'est-à-dire au niveau du matériel, des systèmes d'exploitation, du serveur web et des services applicatifs hébergés. Ce dernier point est problématique. En effet, la diversité du matériel, des systèmes d'exploitation et des serveurs web permet de mettre en œuvre une diversification fonctionnelle efficace. Cependant, de plus en plus de vulnérabilités touchent des applications hébergées par les serveurs web. Ces applications, destinées à fournir un service particulier, sont écrites dans un langage déterminé. Leur diversification telle qu'entendue par les auteurs est donc relativement difficile à mettre en œuvre, à moins de faire de la programmation N-versions comme proposé initialement.

2.1.5 Détection d'intrusions par scénarios modulaire

Dans [2], Almgren et *al.* décrivent un outil de détection d'intrusions par scénarios pour surveiller l'activité d'un serveur web, notamment les requêtes vers les scripts hébergés sur le serveur, cibles particulièrement vulnérables. L'outil utilise les fichiers de log générés par le serveur web pour effectuer son analyse. Il peut aussi se positionner en consommateur direct des logs émis par le serveur. De cette façon, l'analyse peut s'effectuer aussi bien en différé ou en continu. L'analyse différée permet de déporter l'analyse sur une autre machine, de sorte que l'activité du serveur web n'est pas affectée par une quelconque consommation de ressources système par l'outil de détection.

L'outil de détection présenté se caractérise par une architecture modulaire. Pour chaque requête, une séquence de traitements propres à chaque module est effectuée afin de détecter l'activité intrusive.

Le premier module initialise le traitement. La ligne de log est découpée suivant les champs du format CLF. Les caractères imprimables encodés au format hexadécimal sont décodés afin d'uniformiser les traitements effectués par la suite et de détecter la présence d'encodages suspects. Si un de ces traitements échoue, de façon partielle ou totale, une première alerte est émise et la structure de données éventuellement incomplète est transmise au module suivant. Le second module recherche des signatures d'attaques ou d'activités licites dans les différents champs de la ligne de log. Les signatures sont des expressions régulières destinées à un champ particulier de la ligne de log. Les signatures utilisées ont ceci de particulier qu'elles permettent de déclencher des alertes relatives à une activité intrusive, mais elles permettent aussi de positionner des indicateurs d'une activité licite. Cette double information concernant la nocivité ou l'innocuité d'une requête permet d'évaluer, par la suite, sa dangerosité globale. Le troisième module utilise les alertes émises par les précédents modules afin de les fusionner et d'émettre éventuellement une nouvelle alerte. Ce traitement permet d'informer l'opérateur de la dangerosité globale d'une requête, alors que les premières alertes émises, considérées séparément, ne permettent pas forcément d'identifier clairement le danger encouru. Le quatrième module utilise lui aussi les alertes précédemment émises par les autres modules. Il peut effectuer des analyses complémentaires de la ligne de log, subordonnées par la présence de certaines alertes dans la structure de données. Ce traitement conditionnel supplémentaire permet tout d'abord une analyse plus poussée de la ligne de log. Il permet ensuite de ne déclencher que des traitements nécessaires et donc d'économiser la consommation de ressources de l'outil. Le cinquième module effectue une forme de veille vis-à-vis des hôtes ayant eu une activité intrusive contre le serveur web. Les adresses IP des clients ayant déjà formulé des attaques contre le serveur web sont listées et toute activité émanant de ces adresses est consignée. Ainsi une requête apparemment saine, c'est-à-dire n'ayant déclenché aucune alerte, provenant d'un client ayant eu une activité intrusive auparavant, sera marquée et pourra être analysée manuellement par l'opérateur de sécurité. Cette activité de veille permet éventuellement de découvrir de nouvelles attaques non décelées avec les signatures utilisées par l'outil. Le sixième module est chargé d'éliminer les alertes liées à une activité connue pour être non intrusive. Par exemple, il n'est pas nécessaire de rapporter les

alertes liées à un audit de sécurité que l'opérateur aurait pu effectuer pour détecter la présence de vulnérabilités sur le serveur web. Le septième module a une activité décisionnelle. Il analyse les alertes associées à la requête initiale afin de déterminer si cette dernière est présentée ou pas à l'opérateur de sécurité. Le but est de hiérarchiser les alertes. Le huitième et dernier module est chargé de la présentation des alertes.

Cet outil de détection se caractérise par une architecture modulaire calquée sur les différentes phases d'analyse effectuées. Cette architecture prend en entrée une ligne de log émise par le serveur web, encapsulée dans une structure de données. Cette structure de données est ensuite analysée et enrichie par les différents modules. L'architecture modulaire de l'outil présente plusieurs avantages. Elle permet tout d'abord de distribuer l'analyse sur plusieurs machines, ce qui, face à des serveurs web ayant une très forte charge, est un avantage. D'autre part, la modularité introduit une souplesse dans la maintenance et l'évolution de l'outil. Un nouveau module peut facilement être ajouté à la chaîne de traitement pourvu qu'il puisse appréhender et enrichir la structure de données créée initialement.

L'enrichissement de la structure de données s'effectue en lui associant, pendant les différentes étapes d'analyse, un certain nombre d'alertes. Les alertes sont affectées en fonction des caractéristiques propres au contenu de la ligne de log initiale et des alertes précédemment associées par un module donné.

Les travaux que nous présentons au Chapitre 5 concernant la mise en œuvre d'un outil de détection d'intrusions par scénarios sont basés sur cette architecture.

Le mode de détection conditionnelle apporté par le quatrième module, semble être un atout, puisqu'il ne semble effectuer que des traitements nécessaires, en fonction de caractéristiques présentes dans la requête effectuée par le client. Cependant, les auteurs ne décrivent pas exhaustivement ce mode de fonctionnement, ni n'apportent de preuve que le traitement conditionnel n'implique pas de faux négatifs. En effet, il est possible que le déclenchement de certains traitements liés à la détection de l'exploitation d'une vulnérabilité ne soit pas effectué alors que cette vulnérabilité est bien exploitée. De plus, le cinquième module, lié au suivi des hôtes ayant eu une activité intrusive, ne semble pas adéquat. En effet, le suivi s'effectue via les adresses IP. Cela est relativement difficile, puisque les clients connectés sur internet changent d'adresse IP régulièrement. Ce module peut induire l'opérateur de sécurité en erreur quant à l'activité à surveiller, et donc occasionne une charge de travail supplémentaire et potentiellement inutile pour celui-ci.

2.1.6 Interaction entre un serveur web et une base de données

Les applications destinées à servir un contenu dynamique aux clients sont souvent couplées à une base de données. Lors de la connexion d'un client, l'application hébergée sur un serveur web interroge une base de données, récupère le contenu spécifique et le met en forme pour le client ou modifie les données propres au client dans la base de données. L'interaction entre les applications hébergées sur le serveur web et la base de données se font en général avec des transactions SQL. Des travaux s'appliquent à surveiller les interactions entre des applications hébergées par le serveur web et d'autres

services, comme des serveurs de base de données.

Les travaux présentés par Lee et *al.* [46] puis par Low et *al.* [52] proposent un mode de détection d'intrusions comportementale visant à reconnaitre les transactions SQL saines. Lors d'une phase d'apprentissage, les auteurs constituent des empreintes de transactions SQL saines. Les requêtes SQL étant générées de façon automatique, elles ont une structure prévisible et régulière. Cette structure est consituée d'invariants tels que les mots clés `SELECT` ou `WHERE` et de parties variables qui sont les paramètres fournis par le client. La constitution d'une empreinte s'effectue autour de ces caractéristiques. La structure de la requête et les invariants constituent le squelette de l'empreinte. Des expressions régulières représentent les parties variables, ce qui complète l'empreinte. Les expressions régulières servent à décrire le type de données ou la forme des données autorisées. Par exemple, une expression régulière peut indiquer qu'un paramètre doit être un nombre entier compris entre 0 et 100. Finalement, afin d'éviter d'avoir trop d'empreintes de transactions à vérifier, ce qui peut ralentir le traitement, les auteurs proposent de factoriser les empreintes ayant des structures communes. Pendant la phase de détection, les empreintes sont confrontées aux transactions entrantes afin de détecter une éventuelle transaction SQL frauduleuse.

Les travaux proposés par Valeur et *al.* [79] ont des caractéristiques similaires aux travaux que nous venons d'évoquer. Cependant, ils n'utilisent pas de factorisation des modèles de requêtes. Leur méthode de spécification associe un ou plusieurs modèles de requête à une utilisation spécifique de l'application hébergée sur le serveur web. Dans ce contexte, une même ressource utilisée avec deux combinaisons de paramètres différentes (indifféremment des valeurs des paramètres) implique de considérer deux applications différentes. Cette méthode permet de ne tester que les modèles de requêtes propres à une exécution déterminée de l'application hébergée sur le serveur web.

Ces travaux permettent de restreindre les transactions entre le serveur web et la base de données aux seules transactions dont la structure est connue. Ils permettent de plus de se prémunir des attaques par injection de code (*SQL injection*) [5].

Même si ce type d'attaque constitue une part non négligeable des dangers qu'encourt un serveur web, les travaux proposés ne permettent pas de garantir la sécurité du serveur web, mais seulement celle des données contenues dans la base de données.

2.2 Combinaisons de méthodes de détection d'intrusions

Les travaux de détection d'intrusions effectués au SRI (*Stanford Research Institute*) depuis les années 1980 combinent à la fois des méthodes de détection d'intrusions comportementale et des méthodes de détection d'intrusions par scénarios. Ces travaux ont donné naissance au prototype IDES [53] (Intrusion Detection Expert System). Le composant de détection d'intrusions comportementale est basé sur une analyse statistique [30]. Il vérifie que le comportement d'un utilisateur correspond à un profil établi et que l'utilisation des ressources système correspond à un mode de fonctionnement normal. Si la différence entre l'utilisation surveillée et le comportement de référence dépasse un certain seuil, une alerte est émise. Le composant de détection d'intrusions

par scénarios utilise un système expert pour représenter les scénarios d'attaques. Ces scénarios correspondent à des exploitations de vulnérabilités connues et des violations de politique de sécurité. Les deux composants utilisent les données d'audit système pour surveiller des stations de travail en temps réel. Ces travaux, en appliquant deux méthodes de détection d'intrusions sur les données d'audit système, permettent à la fois d'identifier des attaques connues, mais aussi de découvrir de nouvelles attaques. Cependant les diagnostics proposés par chacun des deux composants peuvent différer et il est nécessaire de résoudre ces conflits.

La version suivante de ce prototype, NIDES [3] (Next-Generation Intrusion Detection Expert System) intègre un *Resolver*. Le *Resolver* utilise l'analyse faite par les deux composants afin de proposer un diagnostic plus intelligent. Son rôle est de prendre en compte les analyses comportementales et par scénarios d'une même activité et de composer un seul diagnostic à partir de ces deux analyses. Le but est de requalifier les fausses alertes émises par le composant de détection d'intrusions comportementale et de ré-évaluer la sévérité des alertes émises. Selon les besoins exprimés par les auteurs, le *Resolver* est censé être capable de prendre en compte les analyses des deux composants de NIDES, mais ils vont plus loin en proposant qu'une seule et même instance du *Resolver* puisse servir pour un déploiement d'outils de détection d'intrusions NIDES sur un système d'information.

Dans [31], il est proposé d'implémenter le *Resolver* suivant la technologie dite du raisonnement procédural ou PRS (*Procedural Reasoning Technology*) [23]. Cette technologie utilise des représentations de connaissances sous forme procédurale pour ré-évaluer les résultats d'analyse des composants de détection d'intrusions comportementale et par scénarios. Cette technologie repose sur un ensemble de faits (alertes émises par les deux composants, état du système surveillé, changements de privilèges et accès aux fichiers par exemple) et un ensemble de buts à réaliser (émission d'un diagnostic). Un ensemble de scénarios décrivant des conditions à remplir ou des actions à effectuer permet d'atteindre les buts ou de réagir dans des situations spécifiques. A partir d'un ensemble de faits, le *Resolver* est capable, suivant les procédures qu'il connait, d'établir un diagnostic prenant en compte l'analyse des composants de détection d'intrusions comportementale et par scénarios, ainsi que l'état du système.

Selon les auteurs, l'utilisation de cette technologie permet d'obtenir un diagnostic précis et *a priori* fiable. Cependant, la modélisation des comportements et l'écriture des procédures nécessitent une très grande expertise concernant le système surveillé et les applications qu'il héberge. Le niveau de détail et de rigueur requis afin de prendre en compte tous les comportements intrusifs rend l'utilisation de cette technologie extrêmement complexe et contraignante pour un opérateur de sécurité.

En pratique [29], l'implémentation du *Resolver* est relativement simple. Toute activité déclarée intrusive par le composant de détection d'intrusions par scénarios entraine l'émission d'une alerte. Une activité déclarée intrusive par le composant de détection d'intrusions comportementale entraine l'émission d'une alerte si au moins une des conditions suivantes est remplie :

– les précédentes données d'audit concernant l'utilisateur surveillé n'ont pas été déclarées anormales ;

– les précédentes données d'audit concernant l'utilisateur surveillé s'effectuaient dans des conditions de consommation de ressources différentes (*top measures*) ;
– le niveau d'anomalie est une fois et demi supérieur à la dernière alerte émise par le composant de détection d'intrusions comportementale pour cet utilisateur.

Contrairement à la solution proposée dans [31], c'est une solution *ad hoc* qui est implémentée pour résoudre les conflits entre les deux composants où le diagnostic du composant de détection d'intrusions par scénarios prévaut sur le diagnostic du composant de détection d'intrusions comportementale.

La différence entre les objectifs assignés au *Resolver*, les technologies utilisables pour résoudre les conflits entre les composants de détection d'intrusions comportementale et par scénarios, et l'implémentation qui est réellement faite montre bien la difficulté de composer un diagnostic à partir de deux procédures d'analyse différentes en parallèle. En effet, la résolution de conflit entre deux diagnostics opposés, s'appuyant sur deux méthodes de détection d'intrusions différentes, n'offre pas de points de comparaison directs sur lesquels il est possible de s'appuyer pour décider d'un diagnostic final.

2.3 Conclusion

Nous avons présenté dans ce chapitre des travaux relatifs à la détection d'intrusions dédiée aux serveurs web, ainsi que des travaux mettant en œuvre une combinaison de méthodes de détection d'intrusions.

La variété des approches utilisées pour surveiller l'activité des serveurs web montre l'enjeu spécifique que représente ce service dans les systèmes d'information. Elle montre en outre que le problème de l'analyse de l'activité des serveurs web reste une question ouverte en détection d'intrusions.

Les travaux effectués au SRI constituent les travaux les plus aboutis concernant l'utilisation d'une combinaison de méthodes de détection d'intrusions à notre connaissance. Ils montrent l'intérêt d'utiliser les méthodes comportementales et par scénarios conjointement. Cependant, ils montrent aussi la difficulté d'établir un diagnostic issu de deux modes d'analyses différents et à partir de sources de données hétérogènes.

Dans la suite de cette thèse, nous étudions les différentes combinaisons possibles de méthodes de détection d'intrusions comportementale et de détection d'intrusions par scénarios. Nous choisissons la plus opportune pour l'analyse de l'activité spécifique des serveurs web.

Chapitre 3

Combinaison de méthodes de détection d'intrusions

Comme nous l'avons expliqué dans le chapitre d'introduction de cette thèse, la détection d'intrusions sur les systèmes d'information se fait suivant deux méthodes complémentaires, apportant des problématiques différentes : la détection d'intrusions comportementale et la détection d'intrusions par scénarios. Usuellement, après analyse des événements selon l'une ou l'autre de ces méthodes, ces derniers sont déclarés soit sains soit intrusifs. Le diagnostic des méthodes de détection d'intrusions comportementale et par scénarios est illustré à la Figure 3.1.

La détection d'intrusions comportementale a le défaut majeur de n'apporter aucune qualification aux événements qu'elle déclare intrusifs. En effet, la seule information dont dispose le processus de détection d'intrusions comportementale concerne l'activité normale sur le système d'information. Lorsqu'un événement est déclaré sain, l'opérateur de sécurité peut facilement savoir pour quelle raison cet événement est sain en se référant au modèle de comportement sur lequel repose le processus de détection comportementale. Lorsqu'il doit traiter un événement déclaré intrusif, c'est à lui d'enquêter, c'est-à-dire de déterminer quelle est l'attaque en cours. Cette lacune entraîne un surcroît de travail pour l'opérateur et peut l'empêcher de prendre des contre-mesures efficaces pour protéger le système d'information.

La détection d'intrusions par scénarios révèle les attaques contre le système d'information en comparant les événements à une base d'attaques préalablement constituée. Si un événement correspond à une attaque, le processus de détection émet une alerte, sinon l'événement est déclaré sain. Cette méthode de détection repose sur un ensemble d'attaques préalablement connues. Ceci implique que tout événement qui n'est pas reconnu comme une attaque est déclaré sain et ne permet pas de détecter de nouvelles attaques.

Dans ce chapitre nous visons à combiner ces deux méthodes de détection d'intrusions et à tirer le meilleur parti de leur capacité de qualification. Ces deux méthodes formulent un diagnostic complémentaire. Dans notre contexte nous cherchons à améliorer la qualité de diagnostic en détection d'intrusions. Nous utilisons à la fois la méthode

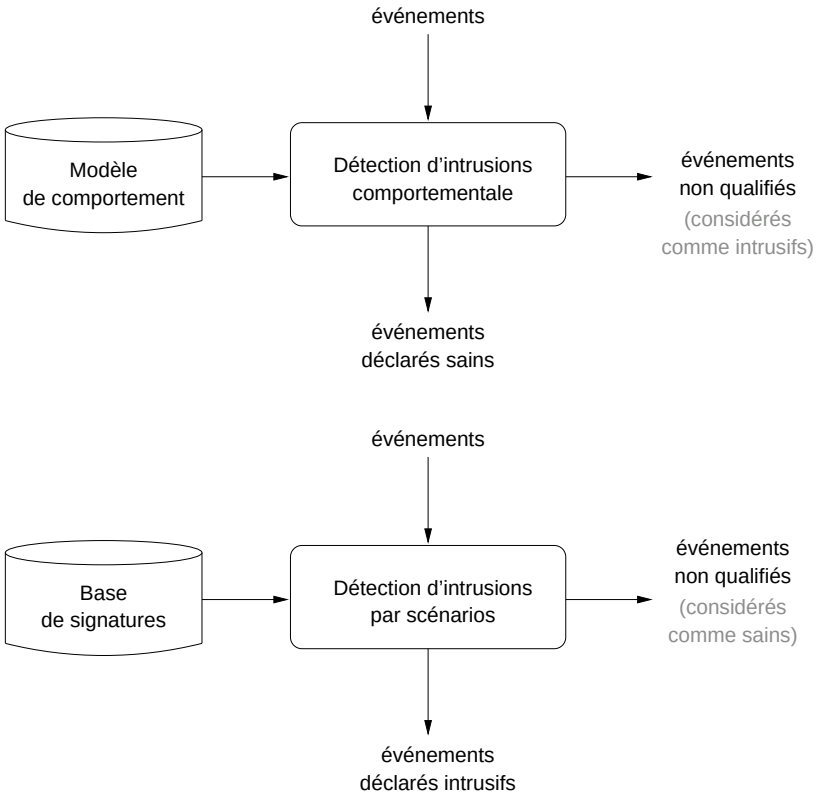

FIG. 3.1 – Diagnostic des méthodes de détection d'intrusions comportementale et par scénarios

de détection d'intrusions comportementale et la méthode de détection d'intrusions par scénarios afin de qualifier complètement l'ensemble des événements à analyser. Pour ce faire, nous présentons tout d'abord, de façon formelle, le diagnostic des méthodes de détection d'intrusions comportementale et par scénarios. Suivant ce formalisme, nous étudions le résultat de la combinaison de ces deux méthodes de détection d'intrusions et choisissons la combinaison la plus adéquate à nos besoins. Finalement, nous discutons des autres combinaisons afin de déterminer quels peuvent être leurs atouts et leurs défauts.

3.1 Formalisation du diagnostic des méthodes de détection d'intrusions comportementale et par scénarios

Afin de comparer les différentes combinaisons, nous nous intéressons à la nature des événements diagnostiqués par les deux méthodes de détection d'intrusions. En effet, aucun outil de détection d'intrusions n'est fiable à 100%. Un outil de détection d'intrusions par scénarios peut, par exemple, émettre une alerte concernant un événement

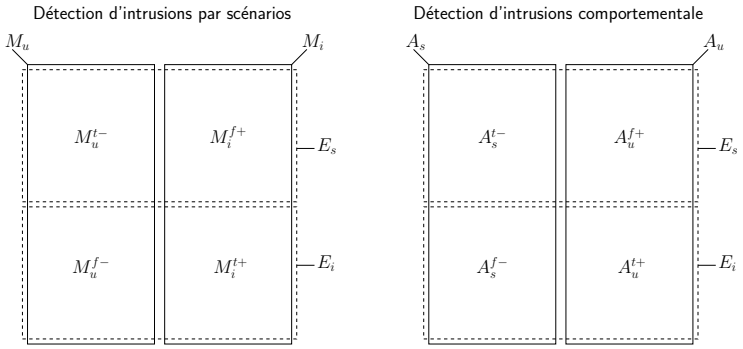

FIG. 3.2 – Formalisation du diagnostic des méthodes de détection d'intrusions en fonction de la nature des événements analysés

sain, ce qui constitue dans ce cas une fausse alerte. Dans cette section, nous définissons des notations permettant de formaliser le diagnostic des méthodes de détection d'intrusions en fonction de la nature des événements analysés. Les définitions sont illustrées dans la Figure 3.2.

Définition 1 ***Événements sains et intrusifs***
Soit E l'ensemble des événements analysés par un outil de détection d'intrusions. E_i est le sous-ensemble de E représentant l'ensemble des événements intrusifs. E_s est le sous-ensemble de E représentant l'ensemble des événements sains.

Un événement ne peut être à la fois sain et intrusif et il ne peut pas être autre chose. Nous avons donc la propriété suivante :

Propriété 1 $E_i \cup E_s = E$ *et* $E_i \cap E_s = \emptyset$

Définition 2 ***Diagnostic de la méthode de détection d'intrusions comportementale***
Soit A_s (resp. A_u) le sous-ensemble de E représentant l'ensemble des événements ***déclarés*** *sains (reps. non qualifiés) par la méthode de détection d'intrusions comportementale.*

Définition 3 ***Diagnostic de la méthode de détection d'intrusions par scénarios***
Soit M_i (resp. M_u) le sous-ensemble de E représentant l'ensemble des événements ***déclarés*** *intrusifs (reps. non qualifiés) par la méthode de détection d'intrusions par scénarios.*

Pour chacune des méthodes de détection d'intrusions, il n'y a que deux diagnostics possibles, exclusifs l'un de l'autre. Nous avons donc la propriété suivante :

Propriété 2 $A_s \cup A_u = E$, $A_s \cap A_u = \emptyset$, $M_i \cup M_u = E$ *et* $M_i \cap M_u = \emptyset$

Les résultats de la détection d'intrusions comportementale et par scénarios ne sont pas parfaits. Nous définissons leurs erreurs et leurs résultats corrects.

Définition 4 ***Résultats de la détection d'intrusions comportementale***
- *Soit A_u^{f+} l'intersection de A_u et de E_s représentant l'ensemble des faux positifs dans le diagnostic de la méthode de détection d'intrusions comportementale,* $A_u^{f+} = A_u \cap E_s$;
- *Soit A_u^{t+} l'intersection de A_u et de E_i représentant l'ensemble des vrais positifs dans le diagnostic de la méthode de détection d'intrusions comportementale,* $A_u^{t+} = A_u \cap E_i$;
- *Soit A_s^{f-} l'intersection de A_s et de E_i représentant l'ensemble des faux négatifs dans le diagnostic de la méthode de détection d'intrusions comportementale,* $A_s^{f-} = A_s \cap E_i$;
- *Soit A_s^{t-} l'intersection de A_s et de E_s représentant l'ensemble des vrais négatifs dans le diagnostic de la méthode de détection d'intrusions comportementale,* $A_s^{t-} = A_s \cap E_s$;

Définition 5 ***Résultats de la détection d'intrusions par scénarios***
- *Soit M_i^{f+} l'intersection de M_i et de E_s représentant l'ensemble des faux positifs dans le diagnostic de la méthode de détection d'intrusions par scénarios,* $M_i^{f+} = M_i \cap E_s$;
- *Soit M_i^{t+} l'intersection de M_i et de E_i représentant l'ensemble des vrais positifs dans le diagnostic de la méthode de détection d'intrusions par scénarios,* $M_i^{t+} = M_i \cap E_i$;
- *Soit M_u^{f-} l'intersection de M_u et de E_i représentant l'ensemble des faux négatifs dans le diagnostic de la méthode de détection d'intrusions par scénarios,* $M_u^{f-} = M_u \cap E_i$;
- *Soit M_u^{t-} l'intersection de M_u et de E_s représentant l'ensemble des vrais négatifs dans le diagnostic de la méthode de détection d'intrusions par scénarios,* $M_u^{t-} = M_u \cap E_s$;

Un événements déclaré sain (resp. intrusif) est soit sain (appartenant à E_s), soit intrusif (appartenant à E_i). Pour chacune des deux méthodes, le diagnostic est soit correct (A_u^{t+}, A_s^{t-}, M_i^{t+}, M_u^{t-}), soit erroné (A_u^{f+}, A_s^{f-}, M_i^{f+}, M_u^{f-}). Suivant les définitions 4 et 5, nous avons la propriété suivante :

Propriété 3

$$\begin{array}{ll} A_u^{f+} \cup A_u^{t+} = A_u & A_u^{f+} \cap A_u^{t+} = \emptyset \\ A_s^{f-} \cup A_s^{t-} = A_s & A_s^{f-} \cap A_s^{t-} = \emptyset \\ M_i^{f+} \cup M_i^{t+} = M_i & M_i^{f+} \cap M_i^{t+} = \emptyset \\ M_u^{t-} \cup M_u^{f-} = M_u & M_u^{t-} \cap M_u^{f-} = \emptyset \end{array}$$

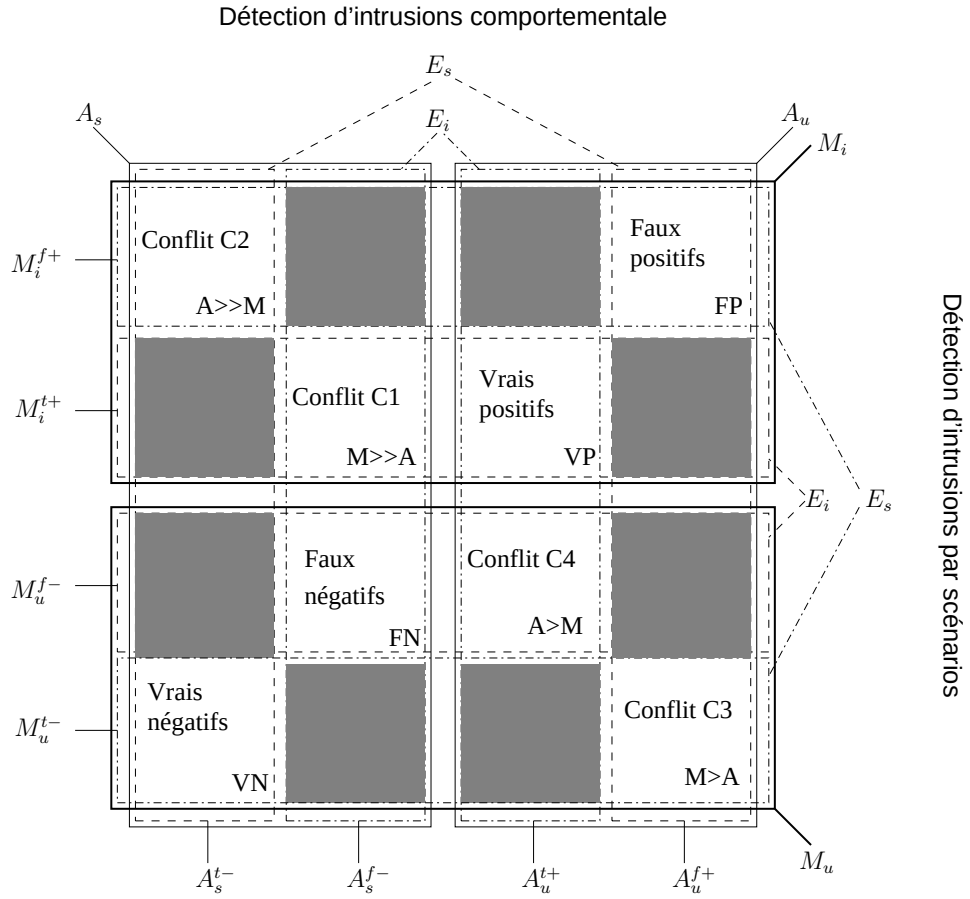

FIG. 3.3 – Combinaison de méthodes de détection d'intrusions comportementale et par scénarios – les ensembles grisés sont des cas de figures impossibles car $E_s \cap E_i = \emptyset$; $X >> Y$ indique que le diagnostic émis par X est correct alors que celui émis par Y ne l'est pas ; $X > Y$ indique que le diagnostic émis par X est plus pertinent que celui émis par Y

3.2 Étude des conflits de diagnostics

L'utilisation de méthodes de détection d'intrusions comportementale et par scénarios permet d'utiliser la capacité de qualification des deux méthodes et donc, d'améliorer la qualité du diagnostic. Cependant, l'utilisation d'une combinaison peut induire des conflits entre les deux diagnostics. La Figure 3.3 représente tous les cas de figure possibles lorsque l'on confronte le diagnostic des méthodes de détection d'intrusions comportementale et de détection d'intrusions par scénarios sur un même ensemble de données.

Dans la suite, nous décrivons les cas où les deux méthodes présentent un diagnostic cohérent. Nous présentons ensuite les conflits. Dans la Figure 3.3, nous représentons le fait qu'une méthode apporte le bon diagnostic par rapport à une autre avec le signe

« >> ». Par exemple, $M >> A$ indique que pour le conflit considéré, la méthode de détection d'intrusions par scénarios diagnostique les événements correctement, contrairement à la méthode de détection d'intrusions comportementale.

Lorsque les événements ne sont qualifiés par aucune des deux méthodes, nous considérons que le conflit est moins grave que lorsqu'il y a une double qualification. Dans ce cas, nous indiquons que l'absence de qualification d'une des deux méthodes est plus opportune que l'autre et nous le notons avec le signe « > ». Par exemple, $M > A$ indique que l'absence de qualification de la méthode de détection d'intrusions par scénarios est correcte, alors que la méthode de détection d'intrusions comportementale aurait dû qualifier ces événements.

- $A_u^{t+} \cap M_i^{t+}$: un événement intrusif n'est pas qualifié par la méthode de détection d'intrusions comportementale et est déclaré intrusif par la méthode de détection d'intrusions par scénarios. Il n'y a pas de conflit, les deux méthodes ont un diagnostic correct. Il s'agit de l'ensemble des vrais positifs, noté VP sur la Figure 3.3.
- $A_s^{t-} \cap M_u^{t-}$: un événement sain est déclaré sain par la méthode de détection d'intrusions comportementale et non qualifié par la méthode de détection d'intrusions par scénarios. Il n'y a pas de conflit, les deux méthodes ont un diagnostic correct. Cet ensemble contient les vrais négatifs, notés VN sur la Figure 3.3.
- $A_u^{f+} \cap M_u^{f-}$: un événement sain n'est pas qualifié par la méthode de détection d'intrusions comportementale et est déclaré intrusif par la méthode de détection d'intrusions par scénarios. Les deux méthodes apportent un diagnostic erroné. Cet ensemble n'est pas conflictuel et contient les faux positifs, notés FP sur la Figure 3.3.
- $A_s^{f-} \cap M_u^{f-}$: un événement intrusif est déclaré sain par la méthode de détection d'intrusions comportementale et non qualifié par la méthode de détection d'intrusions par scénarios. Les deux méthodes apportent un diagnostic erroné. Cet ensemble n'est pas conflictuel et contient les faux négatifs, notés FN sur la Figure 3.3.
- $A_s^{f-} \cap M_i^{t+}$: un événement intrusif est déclaré sain par la méthode de détection d'intrusions comportementale alors qu'il est déclaré intrusif par la méthode de détection d'intrusions par scénarios. Le premier diagnostic est erroné alors que le second est correct. Cet ensemble conflictuel est noté C1 sur la Figure 3.3.
- $A_s^{t-} \cap M_i^{f+}$: un événement sain est déclaré sain par la méthode de détection d'intrusions comportementale alors qu'il est déclaré intrusif par la méthode de détection d'intrusions par scénarios. Le premier diagnostic est correct alors que le second est erroné. Cet ensemble conflictuel est noté C2 sur la Figure 3.3.
- $A_u^{f+} \cap M_u^{t-}$: un événement sain n'est qualifié par aucune des deux méthodes de détection d'intrusions. Le diagnostic de la méthode de détection d'intrusions par scénarios est correct alors que celui de la méthode de détection d'intrusions comportementale ne l'est pas. Cet ensemble conflictuel est noté C3 sur la Figure 3.3.
- $A_s^{t-} \cap M_u^{f-}$: un événement intrusif n'est qualifié par aucune des deux méthodes de détection d'intrusions. Le diagnostic de la méthode de détection d'intrusions comportementale est correct alors que celui de la méthode de détection d'intrusions par scénarios ne l'est pas. Cet ensemble conflictuel est noté C4 sur la Figure 3.3.

Toutes les autres intersections sont vides puisqu'elles sont des sous-ensembles de

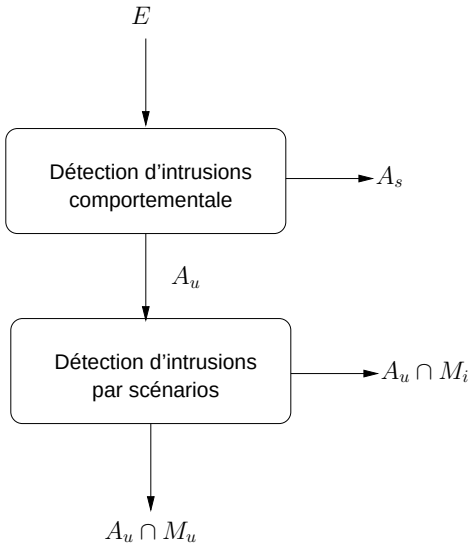

FIG. 3.4 – Combinaison en série $A_u \rightarrow M$: utilisation en série d'une méthode de détection d'intrusions comportementale suivie d'une méthode de détection d'intrusions par scénarios. Les événements non qualifiés par la première méthodes sont analysés par la seconde.

l'intersection $E_s \cap E_i$, qui est vide d'après la propriété 1. Par exemple, nous avons : $A_s^{t-} \cap M_i^{t+} = A_s \cap E_s \cap M_i \cap E_i$.

3.3 La combinaison $A_u \rightarrow M$

Dans cette section, nous présentons la combinaison en série d'une méthode de détection d'intrusions comportementale suivie d'une méthode de détection d'intrusions par scénarios (notée $A_u \rightarrow M$). Nous étudions les conséquences de l'utilisation d'une telle combinaison relativement au formalisme que nous avons présenté dans la Figure 3.3. Nous montrons ensuite que la combinaison $A_u \rightarrow M$ est adaptée à la nature du trafic HTTP.

La combinaison en série $A_u \rightarrow M$ est illustrée par la Figure 3.4. Dans cette combinaison, la méthode de détection d'intrusions comportementale filtre les événements déclarés sains (A_s). Les événements non qualifiés par la méthode de détection d'intrusions comportementale (A_u) sont ensuite analysés par la méthode de détection d'intrusions par scénarios. La combinaison en série $A_u \rightarrow M$ fournit à l'opérateur trois ensembles de résultats : A_s, $A_u \cap M_i$ et $A_u \cap M_u$.

A_s : Les événements présents dans cet ensemble sont, par définition, considérés comme sains par l'opérateur. Pour un résultat efficace, le modèle de comportement doit être construit de telle sorte que l'analyse par la méthode de détection d'intrusions compor-

tementale n'engendre que peu de faux négatifs ($A_s = A_s^{t-} \cup A_s^{f-}$ avec $A_s^{f-} \simeq \emptyset$).

$A_u \cap M_i$ **:** Cet ensemble contient les événements non qualifiés par la méthode de détection d'intrusions comportementale (soit des attaques potentielles) et déclarés intrusifs par la méthode de détection d'intrusions par scénarios. La méthode de détection d'intrusions par scénarios apporte sa capacité à qualifier les événements qui ne sont pas reconnus sains par la méthode de détection d'intrusions comportementale.

$A_u \cap M_u$ **:** Cet ensemble contient les événements qui n'ont été qualifiés ni par la méthode de détection d'intrusions comportementale, ni par la méthode de détection d'intrusions par scénarios. Ces événements sont soit des faux positifs provenant de l'analyse par la méthode de détection d'intrusions comportementale (A_u^{f+}), soit des faux négatifs issus de l'analyse par la méthode de détection d'intrusions par scénarios (M_u^{f-}). Suivant la nature de l'événement, le modèle de comportement ou la base de signature d'attaques doit être mis à jour.

La Figure 3.5 reprend le formalisme présenté à la Figure 3.3 en tenant compte de la nature des événements analysés ainsi que de la combinaison en série des méthodes de détection d'intrusions comportementale et de détection d'intrusions par scénarios illustrée par la Figure 3.4. Les mesures effectuées dans [75] sur le trafic HTTP vers un serveur web académique et un serveur web industriel montrent que celui-ci est majoritairement sain. Sur la Figure 3.5 cela se traduit par le fait que l'ensemble E_s est plus important en proportion que l'ensemble E_i. Cette mise à l'échelle nous permet de ré-interpréter les conséquences liées à la présence d'ensembles conflictuels dans le contexte de la combinaison en série $A_u \rightarrow M$. La Figure 3.6 illustre le diagnostic final qui est proposé par l'utilisation de la combinaison $A_u \rightarrow M$.

Concernant le conflit C1 ($A_s^{f-} \cap M_i^{t+}$), la méthode de détection d'intrusions comportementale émet un diagnostic erroné. Le fait que ces événements ne soient pas analysés par la méthode de détection d'intrusions par scénarios implique que ce conflit est ignoré par la combinaison $A_u \rightarrow M$. Les événements de l'ensemble conflictuel C1 sont des faux négatifs (FN) dans le contexte de la combinaison $A_u \rightarrow M$. L'enjeu de l'approche que nous proposons est de s'assurer que le conflit C1 soit négligeable, à savoir que la méthode de détection d'intrusions comportementale qualifie comme sains le moins possible d'événements intrusifs.

Les événements appartenant à l'ensemble conflictuel C2 ($A_u^{f+} \cap M_i^{f+}$) sont correctement diagnostiqués par la méthode de détection d'intrusions comportementale. Dans le contexte de la combinaison $A_u \rightarrow M$, ces événements ne sont pas analysés par la méthode de détection d'intrusions par scénarios et le conflit C2 est résolu. L'ensemble conflictuel C2 devient alors un ensemble de vrais négatifs (VN).

Dans le contexte de la combinaison $A_u \rightarrow M$, les ensembles conflictuels C3 ($A_u^{f+} \cap M_u^{t-}$) et C4 ($A_u^{t+} \cap M_u^{f-}$) sont toujours présents. Les événements composant ces deux ensembles ne sont qualifiés par aucune des deux méthodes, l'opérateur les considère alors comme un ensemble d'événements non qualifiés (NQ), contenant des événements sains

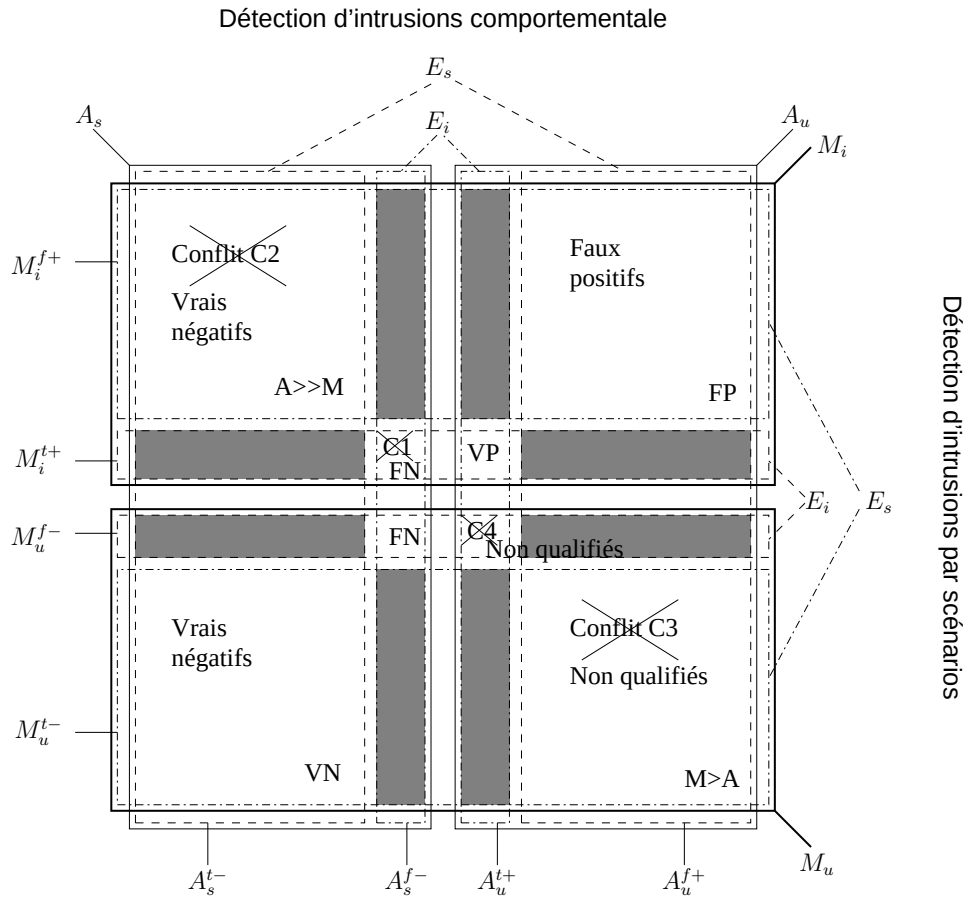

FIG. 3.5 – Diagnostic de la combinaison en série $A_u \rightarrow M$

et intrusifs. La qualification de ces événements requiert une analyse supplémentaire afin de déterminer s'ils sont effectivement sains ou intrusifs, de même que le modèle de comportement de la méthode de détection d'intrusions comportementale et la base de signatures utilisée par la méthode de détection d'intrusions par scénarios doivent être mis à jour.

Dans la combinaison $A_u \rightarrow M$, la méthode de détection d'intrusions comportementale filtre les événements sains qui auraient pu générer de fausses alertes s'ils avaient été analysés par la méthode de détection d'intrusions par scénarios seule. La combinaison $A_u \rightarrow M$ répond donc au problème des faux positifs en détection d'intrusions.

La combinaison $A_u \rightarrow M$ apporte un diagnostic à trois états (sains, intrusifs et non qualifiés)où les événements sont effectivement qualifiés par les deux méthodes mises en œuvre. L'ensemble des événements non qualifiés permet en outre de découvrir de nouvelles attaques en étudiant plus avant les événements qui le composent.

Cependant, la méthode de détection d'intrusions comportementale peut aussi déclarer sains des événements intrusifs (contenus dans les ensembles $M_i^{t+} \cap A_s^{f-}$ et $M_u^{f-} \cap$

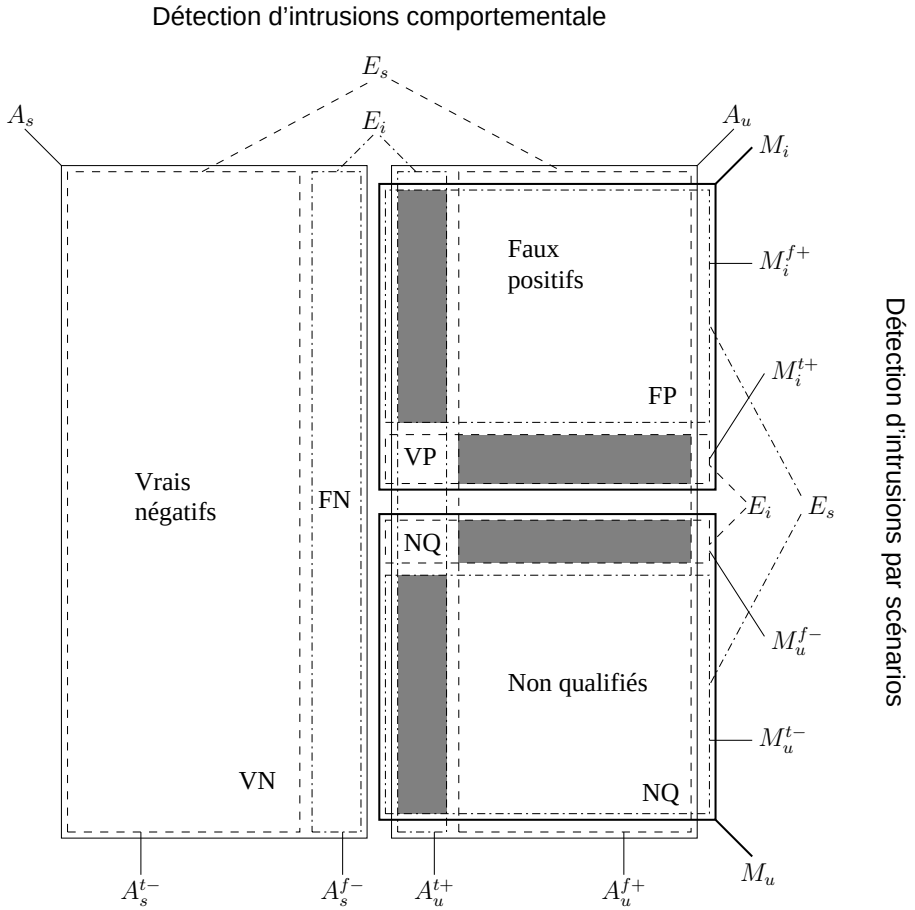

FIG. 3.6 – Diagnostic final de la combinaison en série $A_u \rightarrow M$

A_s^{f-}). L'enjeu de l'utilisation de la combinaison $A_u \rightarrow M$ est de créer un modèle de comportement propre à éviter que les événements intrusifs ne soient filtrés, c'est-à-dire reconnus comme sains, par la méthode de détection d'intrusions comportementale.

3.4 La combinaison $M_i \rightarrow A$

Nous présentons dans cette section une alternative possible à la combinaison $A_u \rightarrow M$ étudiée dans la Section 3.3. Il s'agit d'une combinaison en série d'une méthode de détection d'intrusions par scénarios suivie d'une méthode de détection d'intrusions comportementale (notée $M_i \rightarrow A$). Dans cette combinaison, les événements reconnus dans un premier temps comme intrusifs par la méthode de détection d'intrusions par scénarios sont ensuite analysés par la méthode de détection d'intrusions comportementale. Une telle combinaison a pour but direct d'éliminer les fausses alertes issues de la méthode de détection d'intrusions par scénarios via une requalification des événements

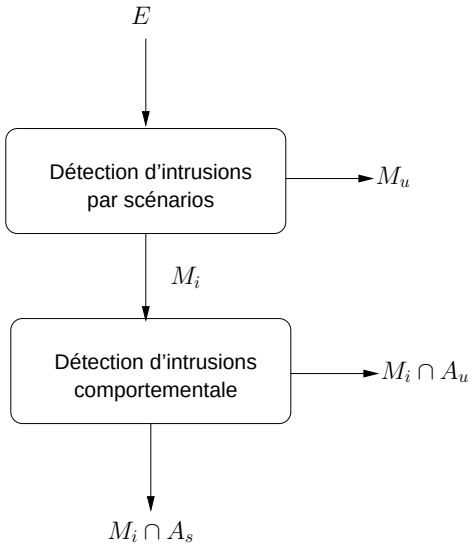

FIG. 3.7 – Combinaison en série $M_i \rightarrow A$: utilisation en série d'une méthode de détection d'intrusions par scénarios suivie d'une méthode de détection d'intrusions comportementale. Les événements reconnus comme intrusifs par la première méthode sont analysés par la seconde.

sains par la méthode de détection d'intrusions comportementale.

Cette combinaison est utilisée dans les travaux présentés par Manganaris et *al.* [58] où les auteurs analysent des flots d'alertes provenant de composants de détection d'intrusions par scénarios avec une méthode de détection d'intrusions comportementale basée sur du *data mining*.

La combinaison $M_i \rightarrow A$ est illustrée par la Figure 3.7. Cette combinaison fournit à l'opérateur trois ensembles de résultats :

- M_u : les événements présents dans cet ensemble de résultat ne sont pas qualifiés par la méthode de détection d'intrusions par scénarios et ne sont pas analysés par la méthode de détection d'intrusions comportementale. Dans la pratique, cet ensemble est considéré comme sain par un opérateur, puisqu'il ne contient pas, selon le diagnostic, d'activité intrusive. C'est le diagnostic qui est fait usuellement lors de l'utilisation d'une méthode de détection d'intrusions par scénarios seule. Pour un résultat efficace, la méthode de détection d'intrusions par scénarios ne doit pas rencontrer d'attaque inconnue par elle afin de ne pas générer de faux négatifs ($M_u = M_u^{t-} \cup M_u^{f-}$ avec $M_u^{f-} \simeq \emptyset$).
- $M_i \cap A_s$: cet ensemble contient les événements tout d'abord déclarés intrusifs par la méthode de détection d'intrusions par scénarios, puis requalifiés comme sains par la méthode de détection d'intrusions comportementale. Les événements de cet ensemble sont considérés par un opérateur comme des faux positifs issus de l'analyse faite par la méthode de détection d'intrusions par scénarios. Pour un

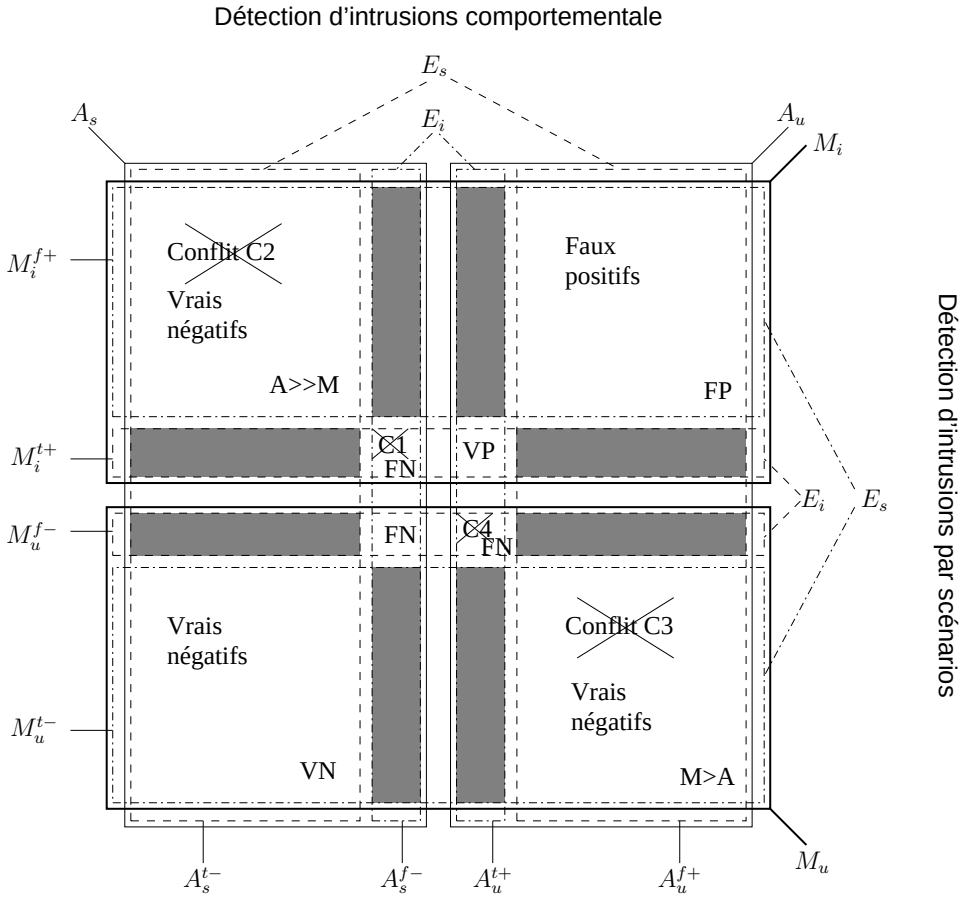

FIG. 3.8 – Diagnostic de la combinaison en série $M_i \rightarrow A$

résultat efficace dans l'élimination de ces faux positifs, le modèle de comportement utilisé par la méthode de détection d'intrusions comportementale ne doit engendrer que peu de faux négatifs ($A_s = A_s^{t-} \cup A_s^{f-}$ avec $A_s^{f-} \simeq \emptyset$).

– $M_i \cap A_u$: cet ensemble contient les événements déclarés intrusifs par la méthode de détection d'intrusions par scénarios et non qualifiés (donc non reconnus comme sains) par la méthode de détection d'intrusions comportementale. Cet ensemble de résultat est considéré comme effectivement intrusif par l'opérateur.

La Figure 3.8 reprend le formalisme présenté à la Figure 3.3 en tenant compte de la nature des événements analysés ainsi que de la combinaison en série des méthodes de détection d'intrusions par scénarios et de détection d'intrusions comportementale illustrée par la Figure 3.7. Comme pour la combinaison présentée à la Section 3.3, cette mise à l'échelle nous permet de ré-interpréter les conséquences liées à la présence d'ensembles conflictuels dans le contexte de la combinaison en série $M_i \rightarrow A$. La Figure 3.9 illustre le diagnostic final qui est proposé par l'utilisation de la combinaison $M_i \rightarrow A$.

Les observations faites à la Section 3.3 concernant les ensembles conflictuels C1 ($A_s^{f-} \cap$

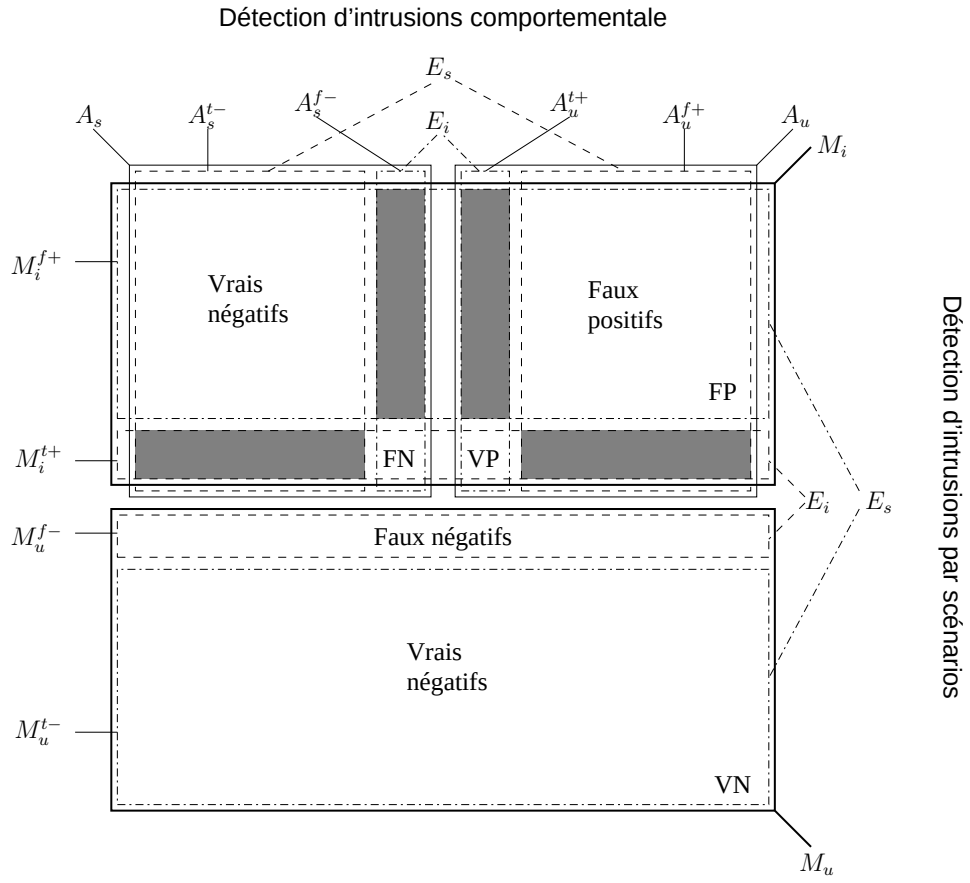

FIG. 3.9 – Diagnostic final de la combinaison en série $M_i \rightarrow A$

M_i^{t+}) et C2 ($A_u^{f+} \cap M_i^{f+}$) sont les mêmes pour la combinaison $M_i \rightarrow A$. Dans le contexte de la combinaison $M_i \rightarrow A$, ces événements sont considérés comme sains. Les événements appartenant à l'ensemble conflictuel C1 deviennent des faux négatifs (FN) et les événements appartenant à l'ensemble C2 deviennent des vrais négatifs (VN).

Concernant le conflit C3 ($A_u^{f+} \cap M_u^{t-}$), le diagnostic émis par la méthode de détection d'intrusions par scénarios est correct. Les événements appartenant à cet ensemble ne sont pas analysés par la méthode de détection d'intrusions comportementale dans le contexte de la combinaison $M_i \rightarrow A$. Ce conflit est ignoré et les événements composant cet ensemble sont des vrais négatifs (VN).

Pour les événements appartenant à l'ensemble conflictuel C4 ($A_u^{t+} \cap M_u^{f-}$), la méthode de détection d'intrusions par scénarios émet un diagnostic erroné. Ces événements ne sont pas analysés par la méthode de détection d'intrusions comportementale, donc ce conflit est ignoré dans le contexte de la combinaison $M_i \rightarrow A$. L'ensemble conflictuel devient un ensemble de faux négatifs (FN). Il apparaît ici qu'aucune nouvelle attaque (c'est-à-dire assez récente pour qu'aucune signature de la base de signature utilisée par

la méthode de détection d'intrusions par scénarios ne puisse la détecter) ne peut être découverte sans une analyse supplémentaire sur les événements composant l'ensemble de résultat M_u.

Dans le contexte de la combinaison $M_i \rightarrow A$, la méthode de détection d'intrusions comportementale est dédiée à la requalification des fausses alertes issues de l'analyse effectuée par la méthode de détection d'intrusions par scénarios. Comme pour la combinaison $A_u \rightarrow M$, la combinaison $M_i \rightarrow A$ répond au problème des faux positifs en détection d'intrusions.

Cependant, la combinaison $M_i \rightarrow A$ ne tire pas complètement partie des capacités de qualification des deux méthodes de détection d'intrusions mises en jeu. Il n'existe pas d'ensemble de résultat qu'un opérateur puisse vraiment considérer comme sain, c'est-à-dire à la fois diagnostiqué sain par la méthode de détection d'intrusions comportementale et ne comportant pas d'événements intrusifs connus selon l'analyse effectuée par la méthode de détection d'intrusions par scénarios. Il en résulte un diagnostic à deux états où un ensemble d'événements non qualifiés par la méthode de détection d'intrusions par scénarios seule doit être considéré comme sain par un opérateur.

3.5 Conclusion

Nous avons présenté dans cette section un formalisme permettant de raisonner sur la combinaison de méthodes de détection d'intrusions. Utilisant ce formalisme, nous avons étudié deux combinaisons en série de méthodes de détection d'intrusions propres à répondre au problème des fausses alertes, et donc destinées à améliorer la qualité du diagnostic en détection d'intrusions.

La combinaison $M_i \rightarrow A$ s'attaque directement au problème des faux positifs en requalifiant les événements déclarés intrusifs par la méthode de détection d'intrusions par scénarios avec une méthode de détection d'intrusions comportementale. Cependant, cette combinaison présente le désavantage de ne pas tirer parti des capacités de qualification des deux méthodes utilisées.

La combinaison $A_u \rightarrow M$ filtre quant à elle les événements qu'elle reconnaît comme sains avec une méthode de détection d'intrusions par scénarios avant que ceux-ci aient pu être déclarés intrusifs à tort par la méthode de détection d'intrusions par scénarios et n'engendrent de fausses alertes. Elle présente à l'opérateur un diagnostic à trois états où les capacités de qualification des deux méthodes de détection sont utilisées. De plus, la présence d'un ensemble d'événements non qualifiés permet de découvrir de nouvelles attaques et de mettre à jour la base de signature ou le mode de comportement suivant la nature des événements.

Suivant les caractéristiques de ces deux combinaisons, nous avons choisi d'utiliser la combinaison $A_u \rightarrow M$ pour analyser du trafic HTTP. Dans ce contexte, la constitution des événements apporte un nouvel argument en faveur de la combinaison $A_u \rightarrow M$. En effet, le trafic HTTP est majoritairement sain. A performance équivalente entre la méthode de détection d'intrusions par scénarios et la méthode de détection d'intrusions comportementale, il vaut mieux utiliser la méthode de détection d'intrusions compor-

tementale en premier lieu car elle reconnaît une majorité des événements et évite ainsi une analyse inutile par la méthode de détection d'intrusions par scénarios.

Chapitre 4

Le module de détection d'intrusions comportementale (*WebFilter*)

Dans cette Section, nous traitons de la mise en oeuvre d'un module de détection comportementale dans le cadre de la combinaison d'outils de détection d'intrusions présentée dans la Section 3. La cible de cette architecture est les serveur web, et nous avons choisi d'utiliser les fichiers de log générés par ces serveurs comme source de données pour l'analyse.

L'objectif du module de détection comportementale, dans le contexte de la combinaison en série, est de reconnaître de façon efficace les requêtes saines qui sont effectuées par un client vers le serveur web. Cette reconnaissance permet d'éviter que des requêtes saines ne soient traitées par le module de détections par scénarios *WebAnalyzer*, et d'alléger ainsi sa charge. *WebAnalyzer* est présenté à le Chapitre 5.

4.1 Source de données pour la détection

Parmi les informations disponibles dans une ligne de log, *WebFilter* utilise uniquement l'URL pour déterminer si la requête est saine ou pas. L'URL est composée du chemin du chemin vers la ressource dans le système de fichier du serveur web ainsi que d'éventuels paramètres.

WebFilter n'utilise pas l'adresse IP fournie dans la ligne de log. En effet, s'il s'agit d'un serveur web accessible depuis Internet, il n'est pas possible d'associer une adresse IP à un client particulier. Par exemple, les fournisseurs d'accès associent à leur client une adresse IP différente à chaque nouvelle connexion sur le réseau. De plus, des clients peuvent se trouver derrière un proxy qui masque leur véritable adresse IP. Finalement, un client mal intentionné peut usurper une adresse IP.

L'aspect temporel n'est pas pris en compte non plus lors de la phase de détection. En effet, un serveur web accessible depuis Internet est censé pouvoir répondre aux clients indifféremment de l'heure où la requête est effectuée. En revanche, nous utilisons

l'aspect temporel lors de la phase de construction du modèle de comportement afin de déterminer le mode d'utilisation qui est fait habituellement par les clients du serveur web.

WebFilter n'utilise pas le code de retour du serveur web lors de la phase de détection. Le code de retour indique si le serveur web a répondu avec succès à la requête ou s'il a dû adopter un autre comportement (redirection, demande d'authentification, erreur). Cependant, même sur une ressource normalement accessible, des erreurs peuvent se produire. En revanche cette information est utilisée lors de la phase de construction du modèle de comportement, là encore pour déterminer le comportement usuel du serveur envers les requêtes qui lui sont envoyées. Il pourrait être plus facile de ne pas soumettre à une analyse les requêtes ayant données lieu à une erreur. En effet, un code de retour indiquant une erreur montre que le serveur n'a pas répondu à la requête du client, il lui a seulement retourné un message d'erreur. Si la requête s'avère être intrusive, l'attaque a échoué. Cependant, nous considérons que même une tentative d'intrusion avortée doit être analysée et signalée à un opérateur.

WebFilter se base sur les ressources demandées et leur mode d'utilisation (présence de paramètres) pour effectuer son analyse. Le modèle de comportement associé à *WebFilter* comporte l'ensemble des ressources saines accessibles sur le serveur web. Les noms des ressources, leur emplacement sur un serveur web et les paramètres qui peuvent leur être associés ne sont pas amenés à changer fréquemment. C'est principalement le contenu des ressources qui est mis à jour. Seule une petite proportion des ressources est supprimée ou ajoutée. Le modèle de comportement contient donc un ensemble de ressources et leur paramètres qui, une fois constitué, ne nécessite que peu de mise à jour. Le processus d'analyse de *WebFilter* ressemble à celui de la consultation d'un dictionnaire. *WebFilter* confronte les requêtes des clients à l'ensemble des ressources présentes dans son modèle. Si la ressource demandée est présente dans le modèle et que la combinaison de paramètres utilisée est correcte, la requête est déclarée saine, sinon elle est analysée par le module de détection par scénarios (*WebAnalyzer*).

Il existe de nombreuses structures de données pour implémenter un dictionnaire. On peut utiliser une simple liste, une table de hachage, un DAG (*Directed Acyclic Graph*), un arbre ou un *trie*[1] [39, 22]. Les ressources proposées par un serveur web sont organisées suivant le système de fichiers qui contient ces ressources. Par exemple, si le site web héberge un forum, les ressources liées au forum sont regroupées dans un répertoire nommé `/forum/`. L'organisation des ressources suivant le système de fichier nous conduit à utiliser une structure arborescente telle qu'illustrée par la Figure 4.1. Nous utilisons une structure similaire à celle présentée dans [62] permettant de stocker et de consulter facilement des données alphanumériques. Les différentes combinaisons de paramètres utilisables avec une ressources sont ajoutées comme feuille de la branche que constitue le chemin de la ressource. Le coût lié au parcours d'une branche, et donc de la comparaison d'une ressource avec cette arborescence, ne dépend pas du nombre d'éléments dans l'arbre représentant le modèle de comportement mais de la profondeur

[1] contraction de *tree* et *retrieval*

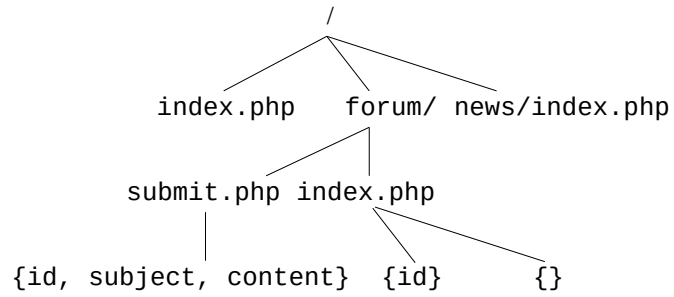

FIG. 4.1 – Représentation des ressources sous forme de *trie*

de l'arbre.

4.2 Le modèle de comportement

Les performances de *WebFilter*, et *a fortiori* les performances et diagnostic de la combinaison $A_u \rightarrow M$, dépendent de la qualité du modèle de comportement. La qualité du modèle s'évalue selon sa complétude et sa correction. La complétude vise à rendre le modèle aussi complet que possible à partir d'un corpus d'apprentissage donné, c'est-à-dire qu'il permette à *WebFilter* de reconnaître le plus d'événements sains possible. La correction est une caractéristique évaluant la qualité du diagnostic de *WebFilter*, notamment sa capacité à ne déclarer sain que des événements effectivement sains.

La construction manuelle du modèle de comportement, à partir d'un corpus d'apprentissage, semble *a priori* bien adaptée aux besoins de complétude et de correction que nous venons d'exprimer. Aucune attaque ne devrait être sélectionnée puisque toutes les requêtes sont analysées par un opérateur de sécurité. La sélection manuelle des ressources nécessite cependant une expertise dans le domaine de la sécurité des applications web ainsi qu'une bonne connaissance des ressources effectivement hébergées sur le serveur. De plus, ce mode de construction s'avère laborieux et très coûteux en temps homme (plusieurs heures) ce qui peut engendrer des erreurs de la part de l'opérateur. Plus le serveur web met de ressources à disposition et plus la fréquentation du site est grande, plus l'opérateur devra extraire, comparer, sélectionner de ressources du corpus d'apprentissage afin de les insérer dans le modèle, afin que celui-ci soit le plus complet et le plus correct possible. Certes, l'opérateur peut utiliser des outils informatiques à sa disposition pour sélectionner les ressources plus facilement, cependant il doit vérifier l'innocuité de chacune des ressources présentes dans le corpus d'apprentissage. Malgré l'investissement en temps et l'expertise nécessaire pour la construction manuelle du modèle, aucune garantie de complétude et de correction n'est apportée.

Pour pallier les défauts liés à la construction manuelle du modèle de comportement, nous proposons d'utiliser une méthode de construction automatisée. L'automatisation de la construction du modèle de comportement a pour but de limiter l'investissement en temps de l'opérateur ainsi que son besoin en expertise technique lors de la création du

modèle. L'automatisation de la construction du modèle de comportement a un impact sur la complétude et la correction du modèle créé. L'utilisation de méthodes d'apprentissages et l'automatisation des choix qui en résulte entraîne des erreurs dans la création de modèle de comportement.En substituant un processus d'apprentissage automatisé à l'expertise d'un opérateur, la correction et la complétude du modèle de comportement deviennent tributaires de la qualité de ce processus et des erreurs qu'il peut engendrer. Nous devons relativiser le gain apporté par l'utilisation d'un tel processus par rapport à la qualité du modèle qu'il fournit.

4.2.1 Méthodologie de construction du modèle comportemental

La méthode que nous proposons est basée sur des méthodes statistiques. Elles sont couramment utilisées par les gestionnaires de sites web afin de se faire une idée du comportement des utilisateurs [34]. Dans ce contexte, l'objectif du gestionnaire du site est d'appréhender les attentes des utilisateurs en étudiant leurs habitudes afin de leur fournir une information plus pertinente, d'améliorer l'accessibilité et la navigation sur le site web. Notre objectif est d'utiliser des méthodes statistiques pour extraire des tendances parmi les accès au serveur web recensés dans le corpus d'apprentissage et faciliter le travail de sélection des ressources par l'opérateur pour la constitution du modèle de comportement.

La procédure proposée consiste à regrouper les ressources selon leurs similarités. Les groupes ainsi formés ont des caractéristiques spécifiques liées à l'ensemble des ressources les composant. Si ces caractéristiques permettent à un opérateur d'évaluer l'innocuité ou la dangerosité potentielle d'un groupe de ressources, alors l'opérateur peut soit sélectionner ou éliminer directement l'ensemble des ressources composants ce groupe, soit analyser de façon plus approfondie ce groupe de ressources.

Le corpus d'apprentissage est un ensemble de fichiers de log où est inscrit l'activité du serveur web pendant une durée déterminée et représentative. A partir de ce corpus, nous associons un ensemble de variables propres à décrire l'activité des clients pour chaque ressource demandée. Les variables utilisées sont décrites dans la Section 4.2.1.3.

Ensuite, nous déterminons le nombre de groupes qu'il est nécessaire de former. En effet, il n'est pas possible de déterminer un nombre de groupes susceptible de convenir à n'importe quel serveur web. De même, il n'est pas possible de former seulement deux groupes, l'un considéré comme sain et l'autre comme contenant des ressources cibles de requêtes intrusives.

Finalement, nous regroupons les ressources suivant les caractéristiques établies pas leur variables, c'est-à-dire suivant leur similarités, et suivant le nombre de groupes que nous avons déterminé.

Parmi les méthodes statistiques existantes, nous avons choisi d'utiliser des méthodes statistiques généralistes et éprouvées telles que la classification ascendante hiérarchique (CAH) et des méthodes d'agrégation autour des centres mobiles.

4.2.1.1 Choix du nombre de groupes

Après extraction des fichiers de log, nous diposons d'un ensemble de ressources R de n ressources, $R = \{r_i, i \in \{1, ..., n\}\}$. Chacune des ressources est représentée par les variables que nous déterminons à la Section 4.2.1.3. C'est à partir de ces variables que nous voulons regrouper les ressources en fonction de leurs similarités.

Nous utilisons la classification ascendante hiérarchique (CAH) afin de déterminer quel nombre de groupe il est nécessaire de former pour un corpus d'apprentissage donné. Initialement, chaque ressource r_i de R constitue un groupe. A chaque étape de l'algorithme, les deux groupes les plus proches sont agrégés et forment un seul et même groupe. Nous obtenons, pour chaque étape de l'algorithme, un nouveau partitionnement composé d'un groupe de moins qu'à l'étape précédente. L'algorithme prend fin lorsque toutes les ressources appartiennent à un seul et même groupe. Les regroupements formés dépendent du critère qui est utilisée pour l'agrégation des groupes. Les critères les plus utilisés sont :

- le critère du "saut minimal" où la distance D entre deux groupes G et G' est la distance entre ses deux éléments les plus proches :

$$D(G, G') = min[d(i, i'), i \in G, i' \in G']$$

- le critère du "saut maximal" où la distance D entre deux groupes G et G' est la distance entre ses deux éléments les plus éloignés :

$$D(G, G') = max[d(i, i'), i \in G, i' \in G']$$

- le critère de la distance moyenne où la distance D entre deux groupes G et G' est la distance moyenne entre leurs éléments :

$$D(G, G') = \frac{\sum_{i \in G} \sum_{i' \in G'} d(i, i')}{|G| \cdot |G'|}$$

- le critère de l'inertie ou critère de Ward. Le critère de Ward consiste, pour chaque étape de l'algorithme, à choisir le partitionnement conservant le plus d'inertie intra-classe (ou faisant perdre le moins d'inertie inter-classes), autrement dit, le partitionnement offrant les groupes les plus compacts. Le lecteur peut se référer à [45] pour une explication plus développée de l'utilisation du critère de Ward.

Chacun de ces critères vise à regrouper les groupes les plus proches à chaque étape de l'algorithme. Le critère d'agrégation que nous utilisons pour notre étude est le critère de Ward car il permet d'obtenir des groupes homogènes et très dissemblables les uns des autres. L'agrégation selon le critère de Ward vise à agréger les groupes en minimisant l'inertie intra-groupe acquise entre deux partitionnements successifs.

Pour chaque étape de l'algorithme, on peut mesurer la nouvelle inertie intra-classes et la reporter dans un histogramme, comme illustré par la Figure 4.2. Nous avons en abscisse le nombre de groupe dans le partitionnement par ordre décroissant et en ordonnée, la mesure d'inertie intra-classes du partitionnement.

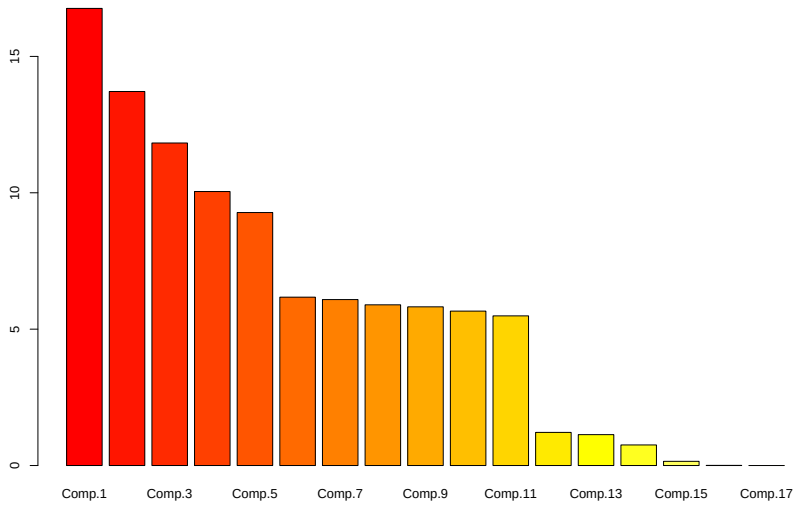

FIG. 4.2 – Exemple d'histogramme des courbes de niveaux ou dendrogramme

Le choix d'un partitionnement se fait en observant une différence d'inertie intra-classes conséquente entre deux partitionnements successifs. En effet, une grande différence d'inertie intra-classes indique l'agrégation de deux groupes très dissemblables qu'il peut être intéressant d'étudier. Par exemple, sur la Figure 4.2, on observe une différence conséquente d'inertie intra-classes entre les partitionnements en 5 et 6 groupes, ainsi qu'entre les partitionnements en 11 et 12 groupes. Il peut donc être intéressant d'étudier les partitionnement en 12 et/ou 6 groupes.

La CAH, en plus de fournir un indicateur sur le nombre de groupes à former, effectue le partitionnement. Cependant, pour agréger deux groupes, cette méthode nécessite de calculer la distance entre chacun des groupes et le groupe nouvellement formé afin de déterminer les groupes les plus proches pour l'agrégation suivante. La CAH effectue autant d'agrégation qu'il y a d'individus à classifier pour déterminer l'ensemble des partitionnements possibles. La complexité de cette méthode est quadratique ce qui la rend difficilement applicable sur un grand nombre d'individus.

Dans la pratique, nous avons trop de ressources à traiter pour pouvoir appliquer la CAH sur l'ensemble de nos données. Nous l'utilisons sur plusieurs échantillons tirés aléatoirement afin qu'elle nous fournisse des indicateurs concernant le nombre de groupes à former.

4.2.1.2 Formation des groupes

Méthode des centres mobiles

La méthode d'agrégation autour des centres mobiles est utilisée pour partitionner un

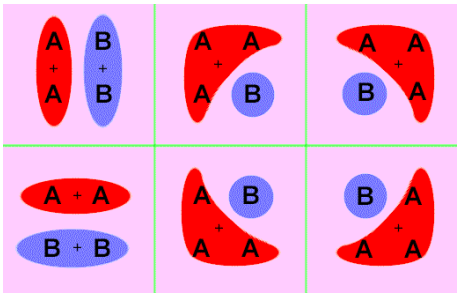

FIG. 4.3 – Exemple de partitionnements possibles en utilisant la méthode d'agrégation autour des centres mobiles

ensemble d'individus en k classes. Les k classes sont déterminés par l'application de plusieurs CAH sur des échantillons tirés aléatoirement de nos données. Cette méthode détermine les k meilleurs centres dans l'espace, c'est-à-dire qu'un centre n'est pas forcément un des individus à classifier. Les individus sont ensuite agrégés autour de ces centres en fonction de leur distance respective. La distance qui est utilisée est la distance euclidienne. Pour déterminer les k centres, la méthode d'agrégation autour des centres mobiles suit les étapes suivantes :

Étape 0 : On détermine aléatoirement k centres de classes provisoires $C_1, ..., C_k$ parmi les individus/ressources de R. Chaque ressource r_i de R est associée à une classe en fonction de sa distance avec chacun des centres de classes $C_1, ..., C_k$. Une ressource r_i appartient à la classe de C_m si elle est plus "proche" de C_m que des autres centres. On obtient une partition $\mathcal{P} = \{P_1, ..., P_k\}$ de l'ensemble des ressources de R.

Étape 1 : On détermine k nouveaux centres de classes $C'_1, ..., C'_k$ qui sont les centres de gravité des partitions $P_1, ..., P_k$. Les centres sont des points de l'espace et non plus des ressources de R. Chaque ressource de R est à nouveau associée à une classe en fonction de sa distance avec les nouveaux centres $C'_1, ..., C'_k$ choisis. On obtient une nouvelle partition $\mathcal{P}' = \{P'_1, ..., P'_k\}$ de l'ensemble des ressources R.

On itère l'étape 1 jusqu'à ce que la partition obtenue soit stable, c'est-à-dire que les partitions formées ne changent plus ou qu'on a effectué un nombre d'itérations fixé au préalable ou que le nombre de réallocations de ressources est passé sous un seuil défini. Pour une preuve que la méthode d'agrégation autour des centres mobiles converge, le lecteur peut se référer à [60].

La méthode des k-moyennes (*k-means*) [59] est une variante de l'algorithme présenté. Les premiers centres sont également tirés aléatoirement parmi les individus. L'étape 1 diffère en ce que les centres sont recalculés à chaque réaffectation d'un individu dans une nouvelle partition. Une seule itération suffit à fournir un partitionnement avec une complexité linéaire. Les résultats de la méthode d'agrégation autour des centres mobiles dépendent des centres qui ont été choisis au départ et ne sont pas reproductibles. Par exemple, la Figure 4.3 illustre les partitionnements possibles sur un ensemble de quatre individus. L'agrégation autour des centres mobiles a aussi le désavantage d'être

particulièrement sensibles aux points aberrants (*outliers*). Un individu éloigné peut déplacer exagérément le centre de gravité d'une partition, ainsi des individus proches ou similaires peuvent être alloués à deux classes différentes. Dans notre cas nous avons ce genre disparité dans nos données. Certaines ressources comme la page d'accueil du site sont très demandées alors qu'une même erreur d'accès due à une mauvaise écriture du chemin vers une ressource n'arrive qu'épisodiquement.

Partitioning Around Medoids (PAM)

Suivant la méthode d'agrégation autour des centres mobiles, le calcul d'un centre de gravité en dehors des ressource que l'on cherche à classifier peut engendrer un partitionnement qui ne reflète pas réellement les similarités des ressources regroupées. Plutôt que de calculer un centre de gravité, on peut comme centre d'un groupe la ressource la plus représentative de ce groupe [37]. L'élément choisi comme centre d'un groupe est appelé médoïde. Cet élément, le médoïde, est celui possédant la dissimilarité moyenne la plus faible d'avec les autres éléments du groupe. Avec cette méthode, chaque groupe est donc représenté par l'élément le plus représentatif du groupe. La méthode PAM ou *k-medoids* suit les étapes suivantes :

Étape 0 : On fixe aléatoirement k représentants de classes (pour un partitionnement en k groupes) $C_1, ..., C_k$ parmi les ressources de R. On obtient un premier partitionnement $\mathcal{P} = \{P_1, ..., P_k\}$ de l'ensemble des ressources de R ;

Étape 1 : On sélectionne aléatoirement un médoïde C_i et un élément O du groupe i.

Étape 2 : On calcule la nouvelle partition obtenue en inversant O et C_i. Si la qualité de la nouvelle partition est supérieure à la précédente, alors O est la nouvelle médoïde du groupe i ; Une partition est considérée meilleure si on observe une diminution de la distance moyenne entre les éléments et les représentants de leurs classes ;

L'algorithme reprend à l'étape 1 jusqu'à stabilisation.

Cette méthode a pour avantage d'être beaucoup moins sensible aux points aberrants (*outliers*) que la méthode d'agrégation autour des centres mobiles, mais sa complexité, comme pour la CAH, est quadratique, ce qui limite son utilisation sur nos données.

Clustering Large Applications (CLARA)

Afin de pallier le problème de complexité de la méthode PAM, nous utilisons la méthode CLARA [38, 36]. Cette dernière utilise plusieurs échantillons de données afin de déterminer le meilleur partitionnement à effectuer.

La première étape consiste à tirer aléatoirement un échantillon E_1 parmi l'ensemble de nos individus. Il est suggéré [38] d'utiliser un échantillon de $40+2k$ individus, k étant le nombre de groupes à former. En pratique, la puissance de calcul dont nous disposons nous autorise à utiliser des échantillons beaucoup plus volumineux, de l'ordre du millier d'individus. Un fois l'échantillon déterminé, on lui applique la méthode PAM qui fourni k représentant de classes. Tous les individus sont ensuite associés au représentant de classe qui leur est le plus proche. Nous obtenons un premier partitionnement en k groupes.

Ce procédé est reproduit quatre autres fois à partir de quatre autres échantillons de même taille, tirés aléatoirement, E_2, E_3, E_4 et E_5, ce qui donne cinq partitionnements différents.

La qualité d'un partitionnement est déterminée par le calcul de la distance moyenne entre chaque individus et son représentant de classe. Le partitionnement qui est finalement sélectionné est celui ayant la meilleure qualité, soit la distance moyenne de ses individus à son représentant de classe la plus faible.

L'utilisation de la méthode CLARA nous permet de nous affranchir des problèmes de complexité de la méthode PAM. De plus, elle conserve la propriété de la méthode PAM d'éviter une trop grande sensibilité aux *outliers*. Finalement, la puissance de calcul dont nous disposons nous permet d'utiliser des échantillons de données beaucoup plus représentatifs que ceux proposés dans [38]. Dans la pratique, nous n'utilisons la méthode CLARA que s'il s'avère impossible d'utiliser la méthode PAM.

4.2.1.3 Sélection des champs et création des variables

Nous utilisons des fichiers de log dans lesquels sont enregistrées les requêtes faites au serveur web. Les champs composants ces fichiers de log ont été décrits à la Section 4.1. Nous disposons de l'adresse IP du client, de deux champs d'authentification, de l'horodatage de la requête, de la méthode utilisée, du chemin vers la ressource demandée (éventuellement suivie de paramètres), de la version du protocole HTTP utilisé, de la taille des données renvoyées par le serveur et finalement du code de retour du serveur. La ressource est l'objet central de l'analyse. C'est autour de la ressource que l'on va agréger des données statistiques afin de créer des groupes de ressources similaires. Dans cette section nous discutons de la sélection des champs utiles à l'analyse statistique que nous voulons conduire et des variables qui seront créées à partir des champs retenus. Les critères de sélection des champs portent sur la pertinence de l'information liée à chacun de ses champs et sur la façon dont, une fois agrégés pour chaque requête vers une même ressource, ils caractérisent cette ressource.

Pour la suite nous définissons les notations suivantes :

- `Jour` : le nombre de jour de log composant le corpus d'apprentissage ;
- `Req` : le nombre total de requêtes dans le corpus d'apprentissage ;
- $\texttt{Req}_i$: le nombre de requêtes vers la ressource i ;

Les champs non retenus

Parmi les champs disponibles dans les fichiers de log, nous n'utilisons pas l'adresse IP du client, ni la taille des données renvoyées par le serveur.

Adresse IP L'adresse IP est l'identifiant numérique du client sur le réseau. Chaque client a une adresse IP qui est enregistrée dans les fichiers de log lorsqu'il demande une ressource au serveur.

L'adresse IP du client n'est une caractéristique exploitable que si elle apporte une information "topologique" fiable. Or ce n'est pas le cas, puisque une même adresse IP

peut être attribuée de façon "aléatoire" sur un réseau et *a fortiori* sur Internet. Plusieurs clients différents peuvent donc avoir la même adresse IP à des moments différents. De plus une adresse IP peut être usurpée par un client.

L'adresse IP du client n'étant ni un champ fiable, ni une source d'information pertinente, elle n'est pas utilisée pour l'analyse des données.

Taille Pour chaque requête, le serveur web renvoie une certaine quantité de données qui est enregistrée dans les fichiers de log.

La taille des données renvoyées par le serveur web n'est pas seulement liée à la ressource demandée. Cette valeur peut changer si la ressource est dynamique. Le contenu est alors généré à la volée et le volume de données envoyé n'est pas constant d'une requête à une autre ; si la ressource est momentanément indisponible, ce qui peut arriver quelle que soit la qualité du service rendu par le serveur, c'est le volume du message d'erreur qui sera pris en compte. De plus, si la ressource est modifiée sur le serveur, le volume de données renvoyé au client est modifié en conséquence

La taille des données renvoyée ne dépend pas seulement de la consultation qui est faite d'une ressource. Nous ne pouvons pas nous fier à ce champ pour caractériser une ressource.

Les champs retenus et les variables associées

Les champs retenus pour l'analyse sont ceux concernant l'authentification, l'horodatage, la méthode HTTP, la présence de paramètre, la version de protocole et la réponse du serveur. Pour chacun de ses champs, nous créons des variables qui les représentent numériquement. Plusieurs variables peuvent être associées à un même champ. Pour la suite, nous notons $\texttt{Demande}_i$ le nombre de requêtes effectuées faire une ressource i.

Données d'authentification La présence de données d'authentification indique qu'une ressource ne peut être accédée que si le client s'authentifie auprès du serveur. Cette méthode d'authentification devient de plus en plus rare, supplantée par une gestion des authentification au niveau applicatif sur le serveur.

La manifestation d'un besoin d'authentification pour accéder à une ressource est une donnée importante concernant cette ressource, alors que l'identifiant lui-même n'est pas une source d'information intéressante vu le peu d'identifiants observables dans les fichiers de log.

Une variable booléenne ne peut être utilisée car si un client ne s'authentifie pas pour accéder à une ressource le nécessitant, le serveur renvoie un code d'erreur et la ligne de log concernant l'accès à cette ressource ne contient pas d'identifiant. La variable liée au données d'authentification représente pour une ressource i le pourcentage de requêtes qui ont utilisé le mécanisme d'authentification ($\texttt{PctUserId}_i$).

$$\texttt{PctUserId}_i = \frac{\texttt{Req}_i \text{ avec données d'identification}}{\texttt{Req}_i}$$

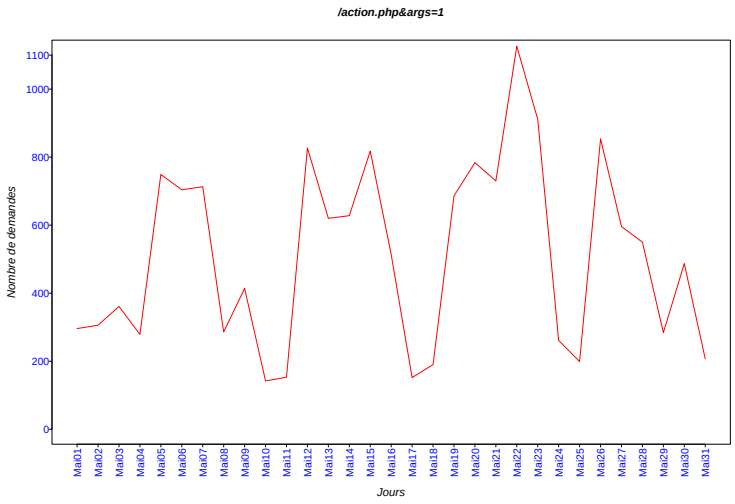

FIG. 4.4 – Observation des requêtes vers la ressource `/action.php` du serveur web de Supélec

Horodatage Pour chaque requête, le serveur enregistre l'heure locale à laquelle elle a été traitée. Ces données se composent de la date, de l'heure (précision à la seconde) et du fuseau horaire où se trouve le serveur.

Des travaux, tel que [83] par exemple, s'intéressent particulièrement à l'aspect temporels des événements afin de trouver et modéliser des "habitudes" et donc par la suite, repérer des anomalies. Cependant ces travaux sont destinés à une analyse purement temporelles des événements. Appliqués à nos données ces travaux peuvent nous fournir des données temporelles concernant les consultations de chaque ressource. Une telle analyse peut difficilement être quantifiée par une variable. De plus, les données temporelles dont nous disposons sont celles du serveur et non pas des clients. Or ces clients peuvent se trouver dans une autre zone horaire que le serveur. De plus, la consultation des ressources d'un serveur peut se faire de façon automatisée, par exemple par des robots indexeurs de pages web, ce qui peut fausser les résultats d'une analyse temporelle.

Une caractéristique notable est cependant la disparité des requêtes que nous observons entre les jours ouvrés et les jours fériés (principalement le week-end). La Figure 4.4 nous montre les requêtes qui ont été effectuées vers la ressource `/action.php` du serveur web de Supélec durant le mois de mai 2003. On observe que des chutes de fréquentation apparaissent périodiquement et correspondent à des jours fériés ou aux *week-end*.

Pour caractériser ce phénomène nous pouvons utiliser les variables $\texttt{PctFérié}_i$ et $\texttt{PctOuvré}_i$.

$$\texttt{PctFérié}_i = \frac{\texttt{Req}_i \text{ les jours fériés}}{\texttt{Req}_i}$$

$$\texttt{PctFérié}_i = \frac{\texttt{Req}_i \text{ les jours ouvrés}}{\texttt{Req}_i}$$

Ces deux variables sont linéairement corrélées ($\texttt{PctFérié}_i = 1 - \texttt{PctOuvré}_i$) et apportent la même information. Dans la pratique nous n'utilisons pour l'analyse statistique que la variable $\texttt{PctFérié}_i$.

Méthode La méthode utilisée par le client indique l'action à mener par le serveur sur une ressource. La méthode `GET` indique que le client veut recevoir le contenu de la ressource. La méthode `POST` indique que le client envoie des informations au serveur dans le corps de la requête. Ces informations sont en général des paramètres que devra traiter la ressource sur le serveur avant de renvoyer des données. Dans ce cas, les informations ne sont pas visibles dans les fichiers de log.

Le protocole HTTP fournit plusieurs méthodes pour effectuer des requêtes vers un serveur et le nombre de commandes peut être étendu par des applications hébergées sur le serveur (pour le protocole WebDAV [24] par exemple). Il est donc difficile d'être exhaustif concernant l'utilisation de ces méthodes. Une observation des fichiers de log montre que les méthodes utilisées en grande majorité sont `GET`, `POST` et `HEAD`.

Pour notre analyse, nous considérons les taux de demandes d'une ressource i avec chacune de ces méthodes en utilisant les variables $\texttt{PctGet}_i$, $\texttt{PctPost}_i$, et $\texttt{PctHead}_i$. Le taux d'utilisation d'autres méthodes, plus marginales, est regroupée dans la variable $\texttt{PctAutresMeth}_i$.

$$\texttt{PctGet}_i = \frac{\texttt{Req}_i \text{ utilisant la méthode } \texttt{GET}}{\texttt{Req}_i}$$

$$\texttt{PctPost}_i = \frac{\texttt{Req}_i \text{ utilisant la méthode } \texttt{POST}}{\texttt{Req}_i}$$

$$\texttt{PctHead}_i = \frac{\texttt{Req}_i \text{ utilisant la méthode } \texttt{HEAD}}{\texttt{Req}_i}$$

$$\texttt{PctAutresMeth}_i = \frac{\texttt{Req}_i \text{ utilisant une autre méthode}}{\texttt{Req}_i}$$

Paramètres Les ressources dites dynamiques sont des programmes hébergés sur le serveur qui sont capable de générer des données à la volée. Ces programmes peuvent prendre des paramètres qui leurs sont envoyés par le client de deux façons différents. Soit ces paramètres sont directement accolés aux noms de la ressource (ex : `/index.php?id=8856efa4`) auquel cas ils sont visibles dans la ligne de log, soit ils sont envoyés dans le corps de la requête HTTP et nous sont alors invisibles. Dans ce dernier cas, la méthode `POST` doit être utilisée par le client pour que le serveur prenne en compte les paramètres qui sont envoyés.

La présence de paramètres est ici encore une information importante puisqu'elle dénote une ressource dynamique et nous disposons de deux indicateurs sûr de leur utilisation. Nous ne considérons pas les noms des paramètres puisque ceux-ci ne sont pas visibles lors de l'utilisation de la méthode `POST`.

Nous dénotons qu'une ressource i est dynamique en utilisant la variable booléenne $\texttt{args}_i$. Si nous observons qu'une ressource est appelée indifféremment avec et sans

paramètres, nous considérons qu'il s'agit de deux ressources différentes et créons en conséquence deux individus distincts.

Protocole Pour chaque requête effectuée vers le serveur, le client doit indiquer quelle version du protocole HTTP il veut utiliser pour l'échange de données. Il existe trois versions de ce protocole : 0.9, 1.0 et 1.1. L'utilisation des deux dernières versions se manifeste dans les fichiers de log par les mots-clés `HTTP/1.0` et `HTTP/1.1`. L'absence de mention de protocole indique que c'est la version 0.9 qui est implicitement utilisée. Les versions 1.0 et 1.1 sont actuellement les versions courantes utilisées par les clients, proxy et serveur. La version 0.9 est considérée comme obsolète même si les serveurs répondent encore aux clients utilisant cette version du protocole HTTP.

La première variable $\texttt{PctHTTP/1.x}_i$ représente le taux d'utilisation des versions courantes du protocole HTTP pour demander une ressource i. Les autres cas observés comme l'utilisation de la version 0.9 ou des utilisations erronées, sont regroupés dans la variable $\texttt{PctHTTP0.9}_i$.

$$\texttt{PctHTTP/1.x}_i = \frac{\texttt{Req}_i \text{ utilisant la version 1.x du protocole HTTP}}{\texttt{Req}_i}$$

$$\texttt{PctHTTP0.9}_i = \frac{\texttt{Req}_i \text{ n'utilisant pas la version 1.x du protocole HTTP}}{\texttt{Req}_i}$$

Ici encore, la variable $\texttt{PctHTTP0.9}_i$ se déduit de la variable $\texttt{PctHTTP/1.x}_i$. En effet, $\texttt{PctHTTP0.9}_i = 1 - \texttt{PctHTTP/1.x}_i$, donc nous n'utiliserons que la variable $\texttt{PctHTTP/1.x}_i$.

Réponse du serveur La réponse du serveur dénote le résultat de la requête initiée par le client. Ce champ est directement lié au comportement du serveur web pour une ressource donnée. Elle va par exemple nous permettre de voir si une ressource demandée par des clients était effectivement mise à disposition par le serveur.

Comme pour les méthodes, le protocole HTTP[63, 21] fournit un grand nombre de réponses possibles pour le serveur, alors que seul un petit sous-ensemble est effectivement utilisé. Ces réponses sont regroupées par ¡¡famille¿¿. Les requêtes réussie ont une réponse en 2xx, une redirection est signifiée par une réponse en 3xx, un échec occasionne une réponse en 4xx et une erreur côté serveur a pour conséquence une réponse en 5xx. Les xx sont des valeurs numériques permettant de donner au client un diagnostic plus précis sur la requête qu'il a effectué, même si la notion de famille de réponse lui suffit pour savoir si sa requête a échoué ou non.

Nous utilisons cette notion de famille de réponse pour la création de nos variables. Pour une ressource i nous étudions la proportion de chaque type de réponse fournie par le serveur avec les variables $\texttt{Pct200}_i$, $\texttt{Pct300}_i$, $\texttt{Pct400}_i$ et $\texttt{Pct500}_i$.

$$\texttt{Pct200}_i = \frac{\texttt{Req}_i \text{ engendrant une réponse de type 2xx}}{\texttt{Req}_i}$$

$$\texttt{Pct300}_i = \frac{\texttt{Req}_i \text{ engendrant une réponse de type 3xx}}{\texttt{Req}_i}$$

Champs	**Variables**
Données d'authentification	$\texttt{PctUserId}_i$
Méthode	$\texttt{PctGet}_i$, $\texttt{PctPost}_i$, $\texttt{PctHead}_i$, $\texttt{PctAutresMeth}_i$
Paramètres	$\texttt{args}_i$
Protocole	$\texttt{PctHTTP/1.x}_i$
Réponse du serveur	$\texttt{Pct200}_i$, $\texttt{Pct300}_i$, $\texttt{Pct400}_i$, $\texttt{Pct500}_i$
Volumétrie	$\texttt{Req-Jour}_i$, $\texttt{PctReq-Jour}_i$

TAB. 4.1 – Liste des 20 variables retenues pour l'analyse statistique

$$\texttt{Pct400}_i = \frac{\texttt{Req}_i \text{ engendrant une réponse de type 4xx}}{\texttt{Req}_i}$$

$$\texttt{Pct500}_i = \frac{\texttt{Req}_i \text{ engendrant une réponse de type 5xx}}{\texttt{Req}_i}$$

Les variables calculées

Volume de requêtes Le volume de requête vers une ressource est un indice révélant la façon dont est consultée une requête et *a fortiori* des groupes de ressources similaires. La volumétrie nous permet de distinguer les ressources en fonction de leur consultation et d'établir « naturellement » des seuils qu'il aurait été difficile d'établir manuellement.

Pour intégrer l'aspect volumétrique, c'est-à-dire prendre en compte le taux de consultation d'une ressource i, nous utilisons la variable $\texttt{Req-Jour}_i$ qui indique le nombre de requêtes moyen par jour vers une ressource i.

$$\texttt{Req-Jour}_i = \frac{\texttt{Req}_i}{\texttt{Jour}}$$

Proportion de requêtes journalière La proportion de requête journalière vers une ressource données nous indique la part de requêtes faite vers cette ressource parmi l'ensemble des requêtes effectuées vers le serveur web. Le calcul de la moyenne de cette proportion nous informe sur la volumétrie relative d'une ressource, comme avec la variable `Req-Jour`, mais aussi sur une éventuelle « stabilité » dans la consultation d'une ressource, notamment par le calcul de la variance.

Nous utilisons la variable $\texttt{PctReq-Jour}_i$ pour représenter la proportion moyenne journalière de requêtes vers une ressource i.

$$\texttt{PctReq-Jour}_i = \frac{\sum_{j=1}^{\texttt{Jour}} \texttt{Req-Jour}_{i\|j} / \texttt{Req}_{\|j}}{\texttt{Jour}}$$

4.3 Partitionnement et analyse des groupes

Dans cette section nous présentons les premiers résultats de l'application de notre analyse statistique décrite à la Section 4.2.1. Nous proposons ensuite une analyse du

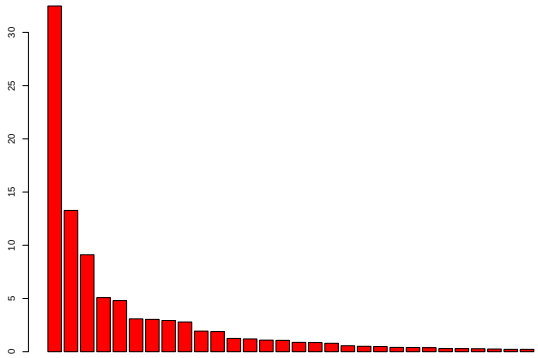

FIG. 4.5 – Histogramme des indices de niveaux pour une CAH réalisées sur un échantillon de 25% des ressources du serveur web de Supélec

partitionnement proposé afin de déterminer des profils de groupes. A partir de ces profils, nous pouvons choisir les groupes que nous voulons intégrer dans un modèle de comportement. De plus, l'analyse des groupes formés peut être réutilisée pour interpréter des partitionnements issus d'autres sources de données, donc d'autres serveurs web. Finalement, nous étudions la composition des groupes formés afin de déterminer la qualité du modèle de comportement créé à partir de notre méthode.

Pour la première application de notre analyse statistique, nous utilisons 31 jours de log du mois de mars 2003 du serveur web de Supélec comme corpus d'apprentissage. Pendant ces 31 jours, le serveur a fonctionné de façon continue, c'est-à-dire sans incident de fonctionnement impliquant un arrêt de service d'une journée ou plus. Ce corpus d'apprentissage contient 866 037 requêtes vers 21 682 ressources (soit environ 40 requêtes par ressource en moyenne et 27 937 requêtes par jour).

4.3.1 Partitionnement des données issus du serveur web de Supélec

Pour choisir le nombre de groupes que nous allons former, nous appliquons une CAH sur les données issues du serveur web de Supélec. Le nombre de ressources à traiter étant trop important pour nos capacités de calcul, nous appliquons une CAH sur un échantillon de nos données tiré aléatoirement. Les ressources n'appartenant pas à l'échantillon sont ensuite regroupées en fonction de leur distance par rapport aux centres des groupes établis. Si l'échantillon sur lequel nous effectuons la CAH comporte des groupes de ressources isolées, le nombre de groupes issus de l'interprétation de la CAH n'est alors pas représentatif de l'ensemble des données que nous voulons partitionner. L'utilisation d'un échantillon tiré aléatoirement peut mener à un partitionnement biaisé. Pour pallier ce risque nous effectuons plusieurs CAH sur plusieurs

échantillons tirées aléatoirement. Pour chaque CAH nous obtenons des résultats sensiblement différents qui peuvent nous donner à choisir entre plusieurs partitionnements. Nous effectuons des CAH sur des échantillons tirés aléatoirement de 25% des données du serveur de Supélec. Les résultats des CAH effectuées nous amènent à considérer un partitionnement des ressources en 6 groupes. La Figure 4.5 nous montre l'histogramme d'indices de niveau sur lequel nous nous sommes appuyés pour choisir le nombre de groupes.

L'interprétation d'un histogramme d'indices de niveaux peut être sujette à caution puisque c'est l'opérateur qui interprète la présence ou l'absence de stabilisation.

Selon la Figure 4.5, une première « stabilisation » apparaît pour les partitionnements en 6, 7, 8, et 9 groupes. Cette stabilisation indique que les groupes qui sont agrégés pour passer d'un partitionnement en neuf groupes à un partitionnement en six groupes sont très similaires. Le dendrogramme montre que le passage à un partitionnement en cinq groupes nécessite d'agréger deux groupes fortement dissimilaires. Nous choisissons d'étudier un partitionnement en six groupes.

Le tableau 4.2 nous montre la répartition des individus dans les six groupes. La première colonne indique le numéro du groupe, la seconde le nombre de ressources composant le groupe et la troisième nous donne la proportion de ressources dans le groupe par rapport au nombre total de ressources. Ce tableau nous montre que la répartition des ressources n'est pas homogène, de même pour le volume de requêtes associé à chaque groupe.

Groupe	**Nb d'individus**	**Pourcentage**	**Nb de requêtes**	**Pourcentage**
1	215	0,99 %	1 051	0,12 %
2	12 751	58,82 %	714 115	82,46 %
3	2 216	10,22 %	74 981	8,66 %
4	4 483	20,68 %	10 014	1,16 %
5	1 628	7,51 %	1 965	0,23 %
6	386	1,78 %	63 911	7,38 %

Tab. 4.2 – Répartition des individus dans les 6 groupes

Les groupes sont présentés plus en détail dans le tableau 4.3. La première colonne représente les variables associées à chaque ressource. Pour chaque groupe, la première colonne représente la moyenne de chaque variable et la seconde la variance calculée à partir des valeurs centrées et réduites des variables. La moyenne nous indique, pour chaque variable, la « tendance » du groupe. La variance nous indique la dispersion des ressources pour chacune de ces variables et permet d'interpréter plus finement la moyenne associée à une variable. Une variance faible pour une variable donnée dénote une forte similarité des ressources pour cette variable. Les variances sont calculées sur les valeurs centrées et réduites des variables afin d'être comparables entre elles.

4.3.2 Étude des valeurs des variables et de leur variances

L'étude des variables que nous avons déterminées à la Section 4.2.1.3 a pour but de déterminer les variables susceptibles d'être à l'origine de certains regroupements et les variables dont le rôle ne semble pas prépondérant. Dans le tableau 4.3, les variables sont regroupées par « familles », séparées par des lignes doubles.

`PctFérié`

La variable `PctFérié` représente le taux de requêtes vers une ressource effectué les jours fériés. Elle nous renseigne directement sur le taux de requêtes vers une ressource effectué les jours ouvrés puisque $\texttt{PctFérié} = 1 - \texttt{PctOuvré}$.

La variance pour la variable `PctFérié` est faible pour les groupes 2, 3, 4 et 5, ce qui indique que les valeurs moyennes associées à cette variable sont assez représentatives des ressources composant ces groupes. Nous observons à partir des moyennes de la variable `PctFérié` que les ressources sont majoritairement demandées les jours ouvrés (entre environ 70% pour le groupe 3 et 99% pour le groupe 4), à l'exception du groupe 5 pour lequel les ressources sont presque exclusivement demandées les jours fériés (99%).

`Req-Jour` − `PctReq-Jour`

Les variables `Req-Jour` et `PctReq-Jour` sont liées à la volumétrie associées à chaque ressource. Les deux variables `Req-Jour` et `PctReq-Jour` sont fortement liées. En effet, `PctReq-Jour` est le taux relatif à la quantité exprimée par la variable `Req-Jour`. La corrélation entre ces deux variables se remarque en observant leurs variances respectives qui sont très proches pour tous les groupes formés. La portée de ces variables dans le regroupement des ressources est difficile à évaluer puisque l'information qu'elles portent est redondante.

Les moyennes observées nous montrent qu'il y a plusieurs niveaux de demandes, notamment matérialisé par les groupes 1 (environ 0,16 demande par jour), 4 (0,07) et 5 (0,4), dont la variance est faible. Cette compacité nous montre que les variables `Req-Jour` et `PctReq-Jour` peuvent être prépondérantes pour le regroupement des ressources vers lesquels il y a peu de demandes. Les autres groupes (2, 3 et 6) semblent moins compacts par rapport à cette variable. Cela peut être relativisé par le fait qu'on observe des volumes de demandes beaucoup plus importants et donc plus « diffus » vers les ressources des groupes 2, 3 et 6. Nous pouvons aussi observer que la variance de ces variables n'est pas liée aux nombre de ressources composant les groupes.

`PctUserId`

La variable `PctUserId` représente le taux de requêtes vers une ressource contenant des données d'identification. Comme la variable `PctFérié`, elle nous renseigne directement sur le taux de requêtes ne contenant pas de données d'identification.

La variable `PctUserId` est en moyenne très faible pour l'ensemble des groupes et nulle pour le groupe 1. Ceci confirme le fait que l'authentification HTTP soit peu utilisée. Les ressources accédées en utilisant l'authentification HTTP semble disséminées au sein des différents groupes (sauf pour le groupe 1). Cette dissémination nous montre qu'il

n'y a pas de regroupement propre à l'utilisation de l'authentification HTTP dans la classification proposée.

`PctGet` − `PctPost` − `PctHead` − `PctVide` − `PctAutresMeth`

Ces variables constituent la « famille » des méthode HTTP utilisées pour effectuer une requête vers une ressource.

Les valeurs de ces variables tranchées pour les groupe 2 à 6. Pour chacun de ces groupes les ressources sont demandées en moyenne avec la méthode `GET` dans plus de 90% des cas. Pour les groupes 2, 4 et 5, la variable `PctGet` est maximale ou presque et la variance des variables relatives à la méthode HTTP utilisée est très faible ce qui dénote une grande compacité de ces groupes. Pour ces mêmes groupes, la variable `PctVide`, qui dénote une utilisation erronée ou suspecte du protocole HTTP, est quasi nulle. Seul le groupe 1 montre une utilisation plus diffuse des méthodes HTTP ainsi que des variances élevées. Ces variables ne semblent pas être un critère de d'agrégation fort pour le groupe 1.

`PctHTTP/1.x`

La variable `PctHTTP/1.x` représente le taux d'utilisation d'une version courante du protocole HTTP pour effectuer des requêtes. Elle nous renseigne sur le taux d'utilisation de version obsolète puisque $\texttt{PctHTTP/1.x} = 1 - \texttt{PctHTTP0.9}$.

Comme pour la variable `PctUserId`, les valeurs de la variables `PctHTTP/1.x` nous montrent que l'utilisation de versions non standard du protocole HTTP (c-à-d autre que les version 1.0 et 1.1) sont marginales et concernent principalement des demandes vers les ressources des groupes 1, 2 et 3. Concernant le groupe 2, même si la valeur de la variable `PctHTTP/1.x` vaut 1, on observe que la variance n'est pas nulle. La valeur de la variable `PctHTTP/1.x` est un arrondi, ce qui signifie que des ressources issues du groupe 2 ont été demandées avec une version du protocole HTTP obsolète.

`Pct200` − `Pct300` − `Pct400` − `Pct500`

Ces variables constituent la « famille » des réponses renvoyés par le serveur aux clients.

Concernant les variables relatives aux réponses fournies par le serveur web aux demandes des clients, nous observons le même phénomène concernant leur valeurs que pour les variables liées aux méthodes HTTP utilisées (`PctGet`, `PctPost`, `PctHead`, `PctVide`, `PctAutresMeth`). Pour les groupe 2 à 6, nous observons qu'une seule variable par groupe à une valeur moyenne de plus de 90%. La variance de ces variables est relativement faible pour les groupes 2 à 5, ce qui dénote une forte compacité relativement à ces variables. Pour le groupe 1, c'est la réponse positive du serveur (*status code* 200) qui domine ; pour les groupes 3, 4 et 5, c'est la réponse négative (*status code* 400) et pour le groupe 6, c'est l'ordre de redirection. Comme pour les variables liées aux méthodes HTTP utilisées, le groupe 1 montre un regroupement plus nuancé avec malgré tout une large majorité pour la réponse négative.

`args`

La variable `args` représente le nombre de ressources dynamiques au sein des groupes. Comme pour les autres variables linéairement corrélées, elle nous renseigne sur le nombre de ressources statiques présente dans les groupes.

Pour la variable `args`, la variance n'a pas d'intérêt puisque cette variable est booléenne et indique seulement si une ressource est dynamique ou non. Seul l'étude de sa valeur moyenne nous renseigne sur les ressources constituant les différents groupes. L'observation de la moyenne de cette variable nous montre qu'il n'y a pas de regroupement propre aux ressources dynamiques. Celles-ci sont disséminées dans les différents groupes.

	Groupe 1		Groupe 2		Groupe 3	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,09	0,749	0,258	0,196	0,314	0,099
Req-Jour	0,158	0,005	1,807	1,357	1,091	0,142
PctReq-Jour	5,119e-06	0,005	6,473e-05	1,366	3,939e-05	0,148
PctUserId	0,0	0,0	0,002	1,395	0,001	0,159
PctGet	0,398	10,418	0,996	0,046	0,975	0,396
PctPost	0,163	79,564	ε	0,002	0,002	0,542
PctHead	0,384	22,960	0,003	0,092	0,022	0,242
PctVide	0,023	98,920	0	ε	0	0
PctAutresMeth	0,033	33,496	ε	0,003	0,001	0,698
PctHTTP/1.x	0,977	81,657	1	ε	0,999	1,707
Pct200	0,137	0,509	0,975	0,016	0,006	0,015
Pct300	0,002	0,032	0,023	0,167	0,001	0,014
Pct400	0,837	0,577	0,001	ε	0,993	0,019
Pct500	0,023	35,343	0,001	0,85	ε	0,699
args	0,172	13,748	0,011	1,065	0,014	1,283

	Groupe 4		Groupe 5		Groupe 6	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,004	0,01	0,99	0,046	0,233	1,287
Req-Jour	0,072	0,003	0,039	ε	5,341	9,809
PctReq-Jour	2,393e-06	0,003	1,811e-06	4,654e-05	1,88e-04	9,482
PctUserId	0,001	0,31	ε	0,206	0,008	4,655
PctGet	1	0,001	1	0	0,947	5,912
PctPost	0	0	0	0	0,001	0,379
PctHead	ε	ε	0	0	0,012	0,408
PctVide	ε	ε	0	0	0,0	0
PctAutresMeth	ε	0,004	0	0	0,04	30,395
PctHTTP/1.x	1	0	1	0	1	0
Pct200	0,001	0,001	ε	0,001	0,052	0,088
Pct300	ε	0,006	0	0	0,946	1,167
Pct400	0,999	0,002	0,999	0,001	0,001	0,001
Pct500	ε	0,346	0	0	0	0
args	0,002	0,236	0,003	0,294	0,005	0,496

TAB. 4.3 – Caractéristiques des 6 groupes résultants d'un partitionnement utilisant la méthode CLARA sur les données provenant du serveur de Supélec. Les variances sont calculées sur les variables centrées et réduites

4.3.3 Étude des groupes

Groupe 2

Le groupe 2 représente un peu plus de la majorité des ressources demandées (58,82%) et les ressources engendrent la quasi-totalité du trafic avec 82,46% des requêtes. Selon le nombre de demandes moyen quotidien (`Req-Jour`) vers les ressources du groupe 2, celui-ci représente environ 23 036 requêtes par jours.

Les ressources du groupe 2 sont quasi-systématiquement demandées avec la méthode `GET` (99,65%). Le taux de réussite des requêtes vers ces ressources est lui aussi quasi total, les réponses de type `200` et `300` représentant 99,85% des réponses.

Ce sont les variables liées aux méthodes HTTP, à la version du protocole et aux réponses du serveur qui sont les plus représentatives de ce groupe.

Groupe 6

Bien que le groupe 6 ne représentent que 1,78% des ressources, les demandes vers les ressources de ce groupe constituent plus de 7% du trafic. Les ressources du groupe 6 sont demandées en moyenne 5,34 fois par jour, ce qui représente environ 1966 requêtes par jour vers les ressources qui composent ce groupe.

À l'image du groupe 2, les ressources du groupe 6 sont quasi-systématiquement demandées avec la méthode `GET` et le taux de réussite de des requêtes vers ces ressources est quasi-total (99,8%). À l'inverse du groupe 2, les requêtes vers les ressources de ce groupe engendrent presque exclusivement des réponses de type `300`. La répartition des requêtes selon les jours ouvrés et jours fériés est aussi du même ordre que celle du groupe 2.

Les variances des variables du groupes 6 nous montrent que ce sont tout d'abord les variables représentant la version du protocole HTTP et les réponses du serveur web qui sont les plus représentatives de ce groupe. Les variables les plus compactes sont ensuite celles relatives à la méthode utilisée.

Groupe 3

Le groupe 3 regroupe environ 10% des ressources demandées et ces ressources engendrent plus de 8% des requêtes ce qui en fait un groupe « équilibré » au regard du volume de ressources et du trafic vers ces ressources. Ce groupe de ressources est demandé, en moyenne, 1,09 fois par jour, ce qui représente un total d'environ 2 419 requêtes par jour en en moyenne, soit environ 8,66% du volume de requêtes que le serveur reçoit.

Comme pour le groupe 2, les ressources du groupe 3 sont quasi-exclusivement demandées avec les méthodes `GET` et `HEAD` (environ 99,67%). Cependant la presque totalité de ces demandes n'ont pu être satisfaites par le serveur web (99,31% pour la variable `Pct400`). La répartition des requêtes entre les jours ouvrés et les jours fériés est du même ordre que celle constatée pour les groupes 2 et 6.

Les variances associées aux variables du groupes 3 nous montrent que ce sont principalement les variables liées à la répartition jours ouvrés/jour fériés, aux réponses du

serveur qui caractérisent les ressources de ce groupe. Viennent ensuite les variables liées aux méthodes utilisées pour demander les ressources.

Groupe 4

Les ressources du groupe 4 sont demandées, en moyenne, environ 0,72 fois par jour, ce qui correspond à environ 322 requêtes par jour. Ces requêtes ne représentent qu'environ 1,15% du volume total des requêtes vers le serveur web, alors que le groupe 4 regroupe plus de 20% des ressources demandées.

Le groupe 4 est très similaire au groupe 3 en ce qui concerne les variables portant sur les méthodes utilisées avec une utilisation quasi exclusive de la méthode `GET`. De même, le serveur web répond presque systématiquement avec un code d'erreur `400`. Les variances associées à ces variables sont même plus faible que celles du groupe 3, ce qui dénote une plus grande compacité du groupe 4. La différence se fait au niveau de la répartition temporelle des requêtes puisqu'elle sont exclusivement faites les jours ouvrés, et au niveau du volume de requêtes faites vers les ressources du groupe 4 qui très faible.

Les variances associées au variables montrent tout d'abord que la méthode utilisée pour demander les ressources et la réponse du serveur sont les variables prépondérantes dans la formation de ce groupe. Les variances nous montrent ensuite que la répartition temporelle des requêtes et le volume de requêtes sont aussi des facteur d'agrégation des ressources.

Groupe 5

Les demandes vers les ressources du groupe 5 se font à hauteur de 0,03894 en moyenne par jour, ce qui correspond, pour l'ensemble du groupe, à environ 63 requêtes par jour. Les requêtes destinées aux ressources du groupe 5 ne représentent que 0,23% du total des requêtes alors que ce groupe représente 7,51% des ressources demandées.

Le groupe 5 est lui aussi très similaire aux groupes 3 et 4 en ce qui concerne les variables relatives à la méthode HTTP utilisée (`GET`) et à la réponse du serveur (`400`). Contrairement au groupe 4, la valeur et la variance de la variable `PctFérié` nous montre que les requêtes vers les ressources du groupe 5 sont presque exclusivement faites des jours fériés.

Comme pour les groupes 2, 3, 4 et 6, ce sont les variables relatives à la méthode HTTP utilisée pour formuler les requêtes et la réponse du serveur qui sont prépondérantes dans la formation de ce groupe. Ces variables ont d'ailleurs des valeurs similaires à celles du groupe 4. La variable qui distingue le groupe 5 du groupe 4 est `PctFérié`.

Groupe 1

Le groupe 1 est le plus petit des six groupes. Les ressources de ce groupe « génèrent » chacune, en moyenne, 0,15769 requêtes par jour, soit environ 34 requêtes par jour pour l'ensemble du groupe.

Les ressources composant le groupe 1 sont principalement demandées avec les méthodes `GET` et `HEAD` (environ 79% des demandes). Les demandes vers les ressources de

ce groupe ont échouées dans environ 83% des cas. Selon le nombre de demandes moyen par jour effectué vers ces ressources, le groupe 1 représente un volume d'environ 34 requêtes par jour, soit environ 0,12% du volume de demandes moyen journalier vers le serveur de Supélec.

Malgré le peu de ressources composant ce groupe (215), on observe une variance très forte pour les variables relatives aux méthodes HTTP utilisées pour accéder aux ressources ce qui dénote une grande dispersion pour ces variables. Il en va de même pour la version de protocole utilisée et la présence d'argument. Les variances les plus faibles, qui dénotent une similarité plus grande, sont celles des variables relatives à la volumétrie et aux réponses du serveur web.

4.3.4 Profils liés aux groupes

Après avoir étudié les variables et les groupes formés, nous pouvons dégager des profils de groupes. Ces profils nous permettront ensuite d'interpréter les groupes ainsi formés et de les sélectionner ou non pour inclusion dans le modèle comportemental.

L'étude des variables et des groupes nous montre que les variables prépondérantes dans la formation des groupes est le couple formé par les variables relatives à la méthode HTTP utilisée et à la réponse du serveur. Nous observons en effet des valeurs quasi maximales pour les groupes 2, 3, 4, 5 et 6. À partir de cette observation, nous pouvons établir deux premiers profils de groupe. Ceux dont les ressources sont demandées avec la méthode `GET` et engendrant une réponse positive du serveur (`200` ou `300`) et ceux dont les ressources sont demandées avec la méthode `GET` et engendrant une erreur (`400`). Le premier profil correspond aux groupes 2 et 6, le second aux groupes 3, 4 et 5. Il est difficile d'associer le groupe 1 à un de ces profils, les valeurs des variables relatives à la méthode HTTP et aux réponses du serveur sont beaucoup plus équilibrées que les variables des autres groupes. Nous pouvons aussi le considérer comme un groupe « poubelle » pour ces variables, ce qui constitue un profil particulier.

De même, nous observons pour la variables `PctFérié` deux tendances qui nous servent pour établir deux autres profils. La premier profil concerne les groupes dont les ressources sont demandées de façon équilibrée. Les ressources des groupes 2, 3 et 6 sont demandées en majorité les jours ouvrés (entre environ 70 et 75% des cas). Le second profil est constitué des groupes dont les ressources sont demandées de façon « tranchées », c'est-à-dire que les requêtes vers les ressources d'un groupe se font quasi-exclusivement, soit les jours ouvrés, soit les jours fériés. Ce second profil correspond aux groupes 1, 4 et 5 : les groupes 1 et 4 regroupent les ressources quasi exclusivement demandées les jours ouvrés, le groupe 5 regroupe les ressources demandées les jours fériés.

Les variables relatives au volume de requêtes vers les ressources, `Req-Jour` et `PctReq-Jour`, peuvent aussi servir à établir des profils bien que les valeurs de ces variables ne soient pas aussi tranchées que celles des variables discutées juste avant. Il est cependant possible de déterminer le profil d'un groupe en fonction du volume de requêtes qui lui est associé. Si on considère les volumes de requêtes qui sont du même ordre de grandeur, nous pouvons dégager trois tendances : le groupe 6 dont les ressources engendrent le plus de requêtes (5,341 requêtes par jour et par ressource) ; les groupes 1, 4 et 5, qui

Profils	**Nom**	**Groupes**
Méthode + Réponse	`GET` + succès	2, 6
	`GET` + échec	3, 4, 5
	« poubelle »	1
Répartition	équilibrée	2, 3, 6
	tranchée	1, 4, 5
Volumétrie	forte	6
	moyenne	2, 3
	faible	1, 4, 5

TAB. 4.4 – Profils des groupes issues de la classification

engendrent le plus faible volume de requête (entre 10^{-1} et 10^{-2} requêtes par jour et par ressource) ; les groupes 2 et 3 qui engendrent un volume de requêtes (entre 1 et 2 requêtes par jour et par ressource) entre ceux du groupe 6 et des groupes 1, 4 et 5.

Le tableau 4.4 récapitule les différents profils que nous avons pu déterminer à partir des caractéristiques des groupes. La première colonne indique le type du profil, c'est-à-dire les variables sur lesquels il repose. La seconde colonne est le nom du profil, fondé sur la valeur des variables sur lesquelles se base le profil. La dernière colonne représente les groupes appartenant à un profil donné, subdivisée en fonction des valeurs des variables de ces groupes.

4.3.5 Interprétation et choix des groupes pour l'intégration au modèle de détection comportementale

Dans cette partie nous interprétons les groupes de ressources selon les profils que nous avons dressés au paragraphe précédent. Cette interprétation nous mène à la sélection et au rejet de groupes de ressources pour le modèle comportementale de notre architecture de détection d'intrusions. La sélection ou le rejet des groupes se fait selon deux critères. Le premier est l'interprétation de l'innocuité des ressources constituant les groupes. Le second est le taux de filtrage que l'intégration d'un groupe dans le modèle de comportement permet.

Les groupes 2 et 6

Le profil « `GET` + succès » des groupes 2 et 6, correspond à un mode de fonctionnement normal du serveur web où les clients accèdent avec succès aux ressources disponibles sur le serveur web. Les groupes 2 et 6 ont un profil équilibré concernant la répartition des requêtes entre les jours ouvrés et les jours fériés ce qui confirme l'idée d'un fonctionnement normal du serveur web pour ces ressources. La volumétrie des groupes 2 et 6 nous indique que les ressources de ces 2 groupes génèrent près de 90% des requêtes et représentent plus de 60% des ressources demandées au serveur web. Cette forte demande nous indique que ces ressources constituent des accès privilégiés au serveur web, tel que celui qu'offre la page d'accueil du site web, par exemple.

Nous considérons que les groupes 2 et 6 sont constitués de ressources saines et sont propres à être intégrés dans le modèle comportemental de notre architecture de détection d'intrusions. Les trois profils auxquels appartiennent les groupes 2 et 6 nous montrent qu'ils doivent regrouper des ressources effectivement disponibles sur le serveur, que ces ressources sont « connus » puisque souvent demandées et finalement que les requêtes liées à ces ressources semblent « normalement » réparties entre les jours ouvrés et les jours fériés. Le taux de succès des requêtes vers les ressources des groupes 2 et 6 sur la période d'observation nous indique que ce sont des ressources pérennes du serveur web. Nous pouvons donc estimer que le taux de 90% de requêtes dirigées vers ces ressources est une évaluation fiable du taux de filtrage qui sera réalisé par le composant de détection d'intrusions comportemental si nous intégrons les ressources des groupes 2 et 6 au modèle de comportement.

Les groupes 4 et 5

Les groupes 4 et 5 représentent des accès au serveurs web qui se sont soldés par des échecs, ce qui signifie que les ressources de ces groupes n'existent pas sur le serveur ou qu'elles ont eu une durée d'accessibilité très brève. La répartition des requêtes vers les ressources des groupes 4 et 5 entre les jours ouvrés et les jours fériés est extrêmement tranchées ce qui dénote un comportement « anormal » pour les ressources appartenant à ces groupes. La volumétrie des groupes 4 et 5 nous montre que les ressources constituant ces groupes (environ 28% des ressources) génèrent peu de trafic (moins de 2%).

Nous choisissons de ne pas intégrer les groupes 4 et 5 au modèle de comportemental de notre architecture de détection d'intrusions. Les ressources sont demandées en moyenne une à deux fois sur la période d'observation et peuvent correspondre, par exemple, à des erreurs de frappe. Le faible nombre de requêtes vers ces ressources peut expliquer la disparité qu'on observe entre les jours fériés et les jours ouvrés. Les ressources appartenant à ces groupes semblant n'être que des erreurs ponctuelles, nous pourrions les intégrer dans notre modèle comportemental. Rien n'exclus cependant que certaines de ces ressources ne soient des cibles d'attaques échouées. Le faible trafic dirigé vers ces ressources nous amène à considérer le taux de filtrage de ces ressources dans le composant de détection d'intrusions comportemental. Les groupes 4 et 5 représentant 28% des ressources pour seulement 2% des requêtes, le taux de filtrage supposément apporté par l'ajout de ces groupes dans le modèle comportemental ne justifie pas la charge supplémentaire occasionnée.

Le groupe 3

Le groupe 3, comme les groupes 4 et 5 a un profil « `GET` + échec » ce qui nous indique que les ressources demandées n'existent pas sur le serveur web. La répartition des requêtes vers les ressources du groupe 3 est équilibrée, comme pour les groupes 2 et 6, ce qui dénote une demande « normale » sinon continue de ces ressources. Le volume de demandes des ressources est « moyen », comme pour le groupe 2 et semble en rapport avec le nombre de ressources constituant le groupe 3. Le taux de demandes représente environ 10% des ressources pour environ 8,5% des requêtes.

Contrairement aux groupes 2 et 6 et aux groupes 4 et 5, le choix de l'intégration ou non du groupe 3 ne semble pas aussi simple. Plusieurs interprétations sont possibles, le groupe 3 partageant des profils des groupes 2 et 6 (répartition et volumétrie) et des groupes 4 et 5 (échec des requêtes vers les ressources constituant ce groupe).

Si nous reprenons le même type d'interprétation que pour les groupes 2 et 6, nous pouvons considérer que le profil « `GET` + échec » est dû à des erreurs récurrentes. Les erreurs peuvent notamment être liées à des requêtes émises automatiquement par un navigateur web lors de la demande d'une ressource par le client. Par exemple, un client qui demande à voir une page web contenant des images dans son navigateur ne fera effectivement qu'une requête vers cette page web. Le navigateur effectue automatiquement les requêtes nécessaires pour obtenir les images et les afficher. Si une image n'existe pas sur les serveur, la requête du navigateur générera une erreur pour le rapatriement de cette image à chaque demande de la page web concernée. Si nous considérons ces ressources comme de simples erreurs récurrentes, le volume de requêtes qu'elles représentent (8,66% du total) nous pousse à intégrer les ressources du groupe 3 dans le modèle de comportement et permet d'éviter à l'opérateur d'avoir à traiter ces erreurs dans le contexte de la détection d'intrusions.

La répartition et la volumétrie observées peuvent aussi être causées par une activité intrusive automatisée ou au moins récurrente (par exemple une activité de type « ver » ou des *scans* de vulnérabilités). Le profil « `GET` + échec » peut alors s'expliquer par le fait que le serveur web n'est pas vulnérable aux attaques subies. La sélection ou le rejet de telles ressources pour notre modèle de comportement ne dépend que des objectifs de l'opérateur de sécurité qui supervise l'architecture. Soit il considère que ces attaques échouées ne sont que des faits bénins et ne sont que des fausses alertes, soit il veut inspecter ces requêtes afin de remonter vers les sources potentielles de ces attaques.

Nous choisissons de sélectionner les ressources du groupe 3 dans le modèle de comportement pour plusieurs raisons. Tout d'abord, le nombre de ressources composant ce groupe (2216) ne semble pas en rapport avec le nombres d'attaques différentes susceptibles de viser un serveur web. Ensuite, même si le groupe 3 contient des ressources cibles d'attaques, le taux d'erreur nous montre que le serveur web n'y est pas vulnérable. Ces attaques peuvent être considérées comme génératrices de fausses alertes. Finalement, le serveur web étant accessible depuis Internet, il n'est pas possible pour un opérateur de déterminer effectivement la source d'une attaque.

Le groupe 1

Le groupe 1 possède les mêmes profils que les groupes 4 et 5 concernant la répartition (tranchée) et la volumétrie (faible). Nous ne pouvons pas déterminer de profils concernant le couple « méthode + réponse » comme pour les autres groupes. Même si le taux d'erreur est important pour ce groupe de ressources (plus de 80%), le taux d'accès réussis doit être pris en compte dans notre interprétation. Le taux d'erreur côté serveur (`Pct500`) est lui aussi sujet à caution à cause de la forte variance associée à cette variable. De même, les résultats concernant les méthodes utilisées pour accéder aux ressources du groupes 1 ne sont pas tranchés comme pour les autres groupes et la forte variance de ces variable limite notre capacité d'interprétation.

Groupe	**Intégré au modèle**
1	non
2	oui
3	oui
4	non
5	non
6	oui

TAB. 4.5 – Récapitulatifs des groupes intégrés au modèle de comportement

Groupe	**Nb de ressources**	**Nb de ressources cibles d'attaques**
1	215	23
2	**12 751**	**0**
3	**2 219**	**24**
4	4 483	111
5	1 628	286
6	**386**	**0**

TAB. 4.6 – Analyse des ressources composant les groupes

Nous choisissons de ne pas intégrer les ressources du groupe 1 dans le modèle de comportement. Ce choix est tout d'abord motivé par la présence d'environ 35 ressources dynamiques dans ce groupe qui peuvent être des cibles potentielles d'attaques. Les erreurs côtés serveur étant principalement dues à des applications hébergées sur le serveur web qui plantent, soit à cause d'une erreur de fonctionnement légitime, soit à cause d'un emploi malicieux de la part d'un attaquant. Notre second argument concerne la volumétrie du groupe 1. Les 215 ressources de ce groupe ne génèrent que 0,12% du trafic vers le serveur web, ce qui représente un taux de filtrage négligeable comparé au risque pris si nous intégrons le groupe 1 au modèle de comportement.

Le tableau 4.5 récapitule nos choix concernant la sélection des groupes dans le modèle de comportement de notre architecture de détection d'intrusions.

Analyse manuelle du contenu des groupes

Dans cette partie nous évaluons la qualité des groupes que nous avons sélectionnés. Pour ce faire nous analysons manuellement chacun des 6 groupes qui ont été formés par l'analyse statistique et recherchons la présence de cibles d'attaques.

Le tableau 4.6 nous montre le résultat de cette analyse. La première colonne indique le numéro du groupe. La seconde colonne rappelle le nombre de ressources contenu dans chaque groupe. La dernière colonne indique le nombre de ressources susceptibles d'être des cibles d'attaques. Les groupes que nous avons choisi d'intégrer dans le modèle de comportement sont marqués en gras.

Parmi les groupes que nous avons sélectionnés (les groupes 2, 3 et 6), seul le groupe

3 contient des ressources susceptibles d'être des cibles d'attaques. Comme l'indique le tableau 4.3 le taux d'échec des requêtes vers les ressources du groupe 3 n'est pas nul. Après analyse, il s'avère que les requêtes réussies ne concernent que des ressources saines. Les ressources cibles d'attaques sont relatives aux vers Nimda[2] et Slapper[3] et à des *scans* vers le script CGI `FormMail`. Si on considère que le filtrage des requêtes vers les ressources cibles d'attaques du groupe 3 engendre des faux négatifs, après analyse, le taux de faux négatifs est évalué à 0,05% des requêtes qui seront filtrées par le module comportemental. Ce qui nous donne, pour le modèle de comportement créé avec notre méthode statistique, un taux de correction de 99,5 %.

Le taux de complétude concerne le taux de ressources saines sélectionnées dans le modèle de comportement, relativement au nombre total de ressources saines présentes dans le corpus d'apprentissage. A partir des chiffres du Tableau 4.6 nous calculons que l'application de notre méthode statistique nous conduit à un taux de complétude de 72,19%. Ce chiffre peut paraître faible, mais il faut le mettre en relation avec le nombre de requête qui sont potentiellement filtrées si le comportement effectif des clients du serveur web correspond au comportement observé dans le corpus d'apprentissage. Ainsi, nous pouvons évaluer le taux de filtrage apporté par les ressources des groupes 2, 3 et 6 grâce au tableau 4.2. Si les ressources sont effectivement pérennes, nous pouvons estimer le taux de filtrage du module de détection d'intrusions comportementale à plus de 98%.

4.4 Conclusion

Dans ce chapitre, nous avons détaillé le module de détection d'intrusions comportementale, *WebFilter*, qui est utilisé dans la combinaison en série $A_u \rightarrow M$ que nous voulons mettre en œuvre. Nous avons proposé un procédé utilisant des méthodes statistiques connues et éprouvées pour construire le modèle de comportement de *WebFilter*. Ce procédé nous a amené à partitionner un ensemble de ressources issues d'un corpus d'apprentissage. A partir des différents groupes, nous avons pu déterminer différents profils qui ont permis de sélectionner des groupes de ressources à intégrer dans le modèle de comportement.

L'utilisation de ce procédé permet un gain de temps conséquent comparé à une analyse manuelle des ressources présentes dans le corpus. De plus, l'expertise nécessaire à l'interprétation des différents groupes est moindre comparé à une connaissance complète de l'architecture et du contenu des ressources présentes sur le serveur web. Notre première expérience montre clairement que le modèle de comportement ainsi créé offre des caractéristiques de complétude et de correction très satisfaisants. De plus, le taux de filtrage attendu sur les requêtes saines émises par les clients du serveur web est presque parfait.

Même si notre procédé peut occasionner des faux négatifs, nous considérons qu'une sélection manuelle des ressources peut elle aussi engendrer des erreurs de sélection de

[2] `http://www.cert.org/advisories/CA-2001-26.html`
[3] `http://www.cert.org/advisories/CA-2002-27.html`

la part de l'opérateur de sécurité, vu leur grand nombre.

Nous n'avons pas discuté de la maintenance, et plus précisément de la mise à jour du modèle de comportement. Cependant, nous considérons que les ressources d'un serveur web ne sont pas amenées à changer régulièrement. En effet, souvent, seul le contenu est mis à jour alors que l'architecture du serveur web, donc les noms des ressources, reste la même. De plus, la mise à jour du modèle de comportement peut facilement se faire via l'analyse des requêtes qui ne sont ni qualifiées par la méthode de détection d'intrusions comportementale, ni par la méthode de détection d'intrusions par scénarios dans notre combinaison en série $A_u \rightarrow M$. Si ces requêtes sont saines, il suffit alors d'ajouter la ressource visée dans le modèle de comportement.

Chapitre 5

Le module de détection d'intrusions par scénarios (*WebAnalyzer*)

Nous avons choisi de développer un outil de détection d'intrusions, *WebAnalyzer*, dédié à la surveillance de l'activité des serveurs web et plus particulièrement à l'interaction entre les clients et le serveur web. La surveillance spécifique des serveurs web réduit le nombre d'attaques auxquelles notre outils est confronté, ainsi que la quantité de données à traiter, comparé à des outils plus « généralistes » telles que les sondes réseau (par exemple Snort [70]). Les objectifs de cet outil sont une précision et une qualité de diagnostic meilleures que celles actuellement fournies par les sondes réseau habituellement utilisées pour surveiller les systèmes d'information.

Nous discutons de ce problème dans la Section 5.1. Le choix de la source de données à utiliser pour effectuer cette surveillance à déjà été établi dans l'introduction (voir Section **??**). Nous développons dans la Section 5.2 le principe de détection de *WebAnalyzer* et le langage de signature utilisé dans la Section 5.3. Nous présentons dans la Section 5.4 quelques exemples de fonctionnement illustrant le diagnostic fourni par *WebAnalyzer*. Finalement, nous discutons de la qualité du diagnostic apporté par *WebAnalyzer* et concluons ce chapitre à la Section 5.5.

5.1 Précision et qualité du diagnostic

La surveillance d'un système d'information est généralement dévolue aux sondes de détection d'intrusions réseaux telles que Snort [70] et Bro [65]. Une disposition judicieuse de sondes réseau permet d'avoir accès à l'ensemble des communications sur le système d'information. L'ensemble des services du système d'information peuvent être surveillés. Le traitement d'un tel volume d'information en temps réel ou presque nécessite cependant de faire des concessions quand à la qualité du diagnostic. Les sondes réseaux doivent être capable de capter l'ensemble du trafic transitant sur le réseau. Ce problème se traite principalement au niveau matériel, tout d'abord en adaptant la ca-

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-CGI count.cgi access"; flow:to_server,established;
uricontent:"/count.cgi"; nocase; reference:bugtraq,128;
reference:cve,1999-0021; reference:nessus,10049;
classtype:web-application-activity; sid:1149; rev:12;)
```

FIG. 5.1 – Signature Snort dédiée à la surveillance des accès au script `count.cgi`

pacité d'écoute de la sonde [16], ensuite en fournissant assez de capacités calculatoires à la machine qui effectue la détection d'intrusions. Les efforts de performances doivent aussi se faire dans le processus de détection lui-même. Le processus de détection doit être aussi rapide que possible afin que le trafic à traiter ne s'accumule pas et que l'opérateur de sécurité puisse prendre, en cas d'attaque, des contre-mesures efficaces dans les meilleurs délais.

La sonde réseau Snort, comme beaucoup d'outils de détection d'intrusions, utilise le *string matching* et le *pattern matching* [10] pour reconnaître les manifestations d'attaques. Ces deux techniques sont relativement coûteuses en temps de calcul et confronter l'ensemble des signatures contenant un motif à chaque événement est impossible si on veut maintenir une détection en temps réel. Snort organise ses signatures sous une forme particulière d'arborescence qui est décrite dans [1]. Le premier niveau concerne le protocole auquel s'applique la signature. Le second niveau filtre les signatures en fonction de l'adresse source et de l'adresse cible de l'événement. Le troisième niveau applique des contraintes de niveau protocolaire telles que la présence d'indicateurs spécifique dans l'entête des paquets, la tailles des paquets ou le sens de la transaction par exemple. En dernier lieu intervient la recherche de motifs. Cette organisation des signatures permet d'éliminer rapidement un grand nombre de signatures à appliquer à chaque événements. Malgré cette organisation, la dernière étape est souvent atteinte, notamment pour surveiller l'activité des serveurs web. Snort dispose de plus de 1 000 signatures dédiées à la surveillance de service web. Dans ce contexte, c'est le motif à chercher lui-même qui détermine le coût de la recherche. Les opérateurs de sécurité s'emploie à écrire des motifs minimalistes, c'est-à-dire des motifs représentant au mieux une attaque ou une vulnérabilité en un minimum d'éléments. Le recours à un motif concis, basé sur une ou plusieurs caractéristiques d'une attaque, risque cependant d'engendrer des fausses alertes, un événement légitime pouvant porter ces caractéristiques.

La Figure 5.1 est une signature issue de la base de signatures de la sonde Snort. Cette signature indique à Snort qu'il doit rechercher le motif `/count.cgi` dans les connexions TCP établies depuis un client quelconque vers le serveur web. Cette signature est la seule signature Snort dédiée à la surveillance du script `Count.cgi` au sein duquel plusieurs vulnérabilités ont été découvertes. La Figure 5.2 présente trois extraits de requêtes HTTP adressées par un client vers le script `count.cgi`. La première requête exploite effectivement la vulnérabilité liée au script en tentant un *buffer overflow* via le paramètre `user`. La deuxième requête est une tentative d'accès de type *scan* destinée à vérifier la présence du script sur le serveur web. La troisième requête est un accès légitime au script `count.cgi`. Chacune de ces trois requêtes contient le motif

```
GET /cgi-bin/Count.cgi?user=a\x90\xbf\xb4\xa0\xff\xbf\xb4\xa0
\xff\xbf\xb4\xa0\xff\xbf\xb4\xa0\xff\xbf\xb4\xa0\xff\xbf\xb4
\xa0\xff\xbf\xb4\xa0\xff\xbf\xb4\xa0\xff\xbf\xb4\xa0\xff\xbf
...
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\xeb<^1\xc0\x89\xf1\x8d
^^X\x88F,\x88F0\x88F9\x88FK\x8dV \x89^V\x8dV-\x89V^D\x8dV1\x89V
^H\x8dV:\x89V^L\x8dV^P\x89F^P\xb0^K\xcd\x801\xdb\x89\xd8@\xcd
\x80\xe8\xbf\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
/usr/X11R6/bin/xterm0-ut0-display0127.000.000.001:0
\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff

GET /cgi-bin/Count.cgi HTTP/1.0

GET /cgi-bin/Count.cgi?df=cfile2|dd=C HTTP/1.0
```

FIG. 5.2 – Trois requêtes HTTP vers le script `count.cgi`

`count.cgi`. La signature « réagit » à chacune de ces requêtes. La sonde de détection d'intrusions émet trois alertes identiques pour ces trois événements différents.

Le problème de la qualité et de la précision du diagnostic se pose ici de trois façons. Nous remarquons tout d'abord qu'avec cette signature, une requête légitime engendre une fausse alerte qui induit l'opérateur de sécurité en erreur. Si on considère le nombre de fausses alertes de ce type qui peuvent être générées, l'opérateur perd beaucoup de temps à traiter des fausses alertes. Ensuite, concernant les requêtes illégitimes, l'opérateur doit inspecter la requête pour qualifier l'attaque, c'est-à-dire déterminer les intentions de l'attaquant. Dans notre exemple une même alerte est associée indifféremment à un simple *scan* et à une tentative de *buffer overflow*. Finalement, l'alerte émise n'indique pas à l'opérateur si l'attaque a réussi ou pas. Ce que nous mettons en exergue ici est le manque de qualité du diagnostic des sondes de détection d'intrusions (fausses alertes) et le manque de précision dans les informations qui sont transmises à l'opérateur. Des travaux ont été effectués visant à pallier ces manques en aval du processus de détection d'intrusions. Ces travaux mettent en œuvre des processus de corrélation afin de réduire les volumes d'alerte, de réduire le nombre de fausses alertes et de présenter à l'opérateur une information synthétique afin qu'il puisse prendre des contre-mesures efficace. A contrario nos travaux visent à exploiter la surveillance spécifique d'un service afin d'améliorer la qualité et la précision du diagnostic directement au niveau de la sonde de détection d'intrusions. Pour se faire nous nous attachons à décrire le plus précisément possible et à contextualiser chaque événement. Nous présentons dans les parties suivantes la solution que nous proposons et les contraintes inhérentes qu'elle impose.

5.2 Principe de détection

Le WebAnalyzer fonctionne comme un consommateur des log fournis par le serveur web. Chaque ligne de log est traitée séquentiellement. La détection s'effectue en

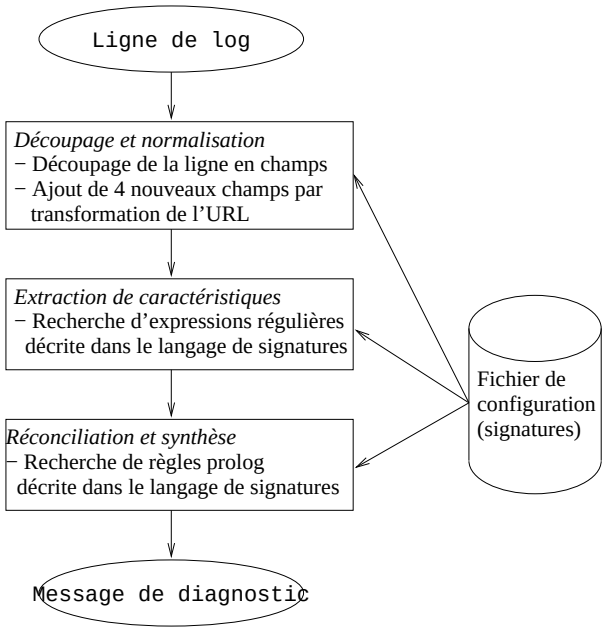

FIG. 5.3 – Architecture des différents modules.

trois phases selon la description faite à la Figure 5.3. La première phase consiste à appréhender la ligne de log et à la préparer pour la détection d'intrusions proprement dite. Lors de la seconde phase *WebAnalyzer* tente de trouver dans la ligne de log des caractéristiques issues des signatures, afin d'y détecter une éventuelle activité malicieuse. La troisième phase consiste à synthétiser les caractéristiques qui ont pu être extraites de la ligne de log afin de proposer un diagnostic précis et pondérer à l'opérateur. Chaque module dispose de son propre jeu de signatures spécifiques. La phase de détection d'intrusions commence donc dès la phase de découpage et normalisation.

5.2.1 Découpage et normalisation

Le module de découpage et normalisation découpe la ligne de log en champs élémentaires tels qu'ils sont listés dans le Tableau 1.1. En cas d'échec du découpage, seuls les champs identifiés sont analysés et une première signature concernant un défaut de lecture est attachée à la ligne de log. Le module de découpage et normalisation n'utilise les signatures présentes dans le fichier de configuration qu'à titre de documentation, celles-ci faisant référence à des erreurs lors de la lecture ou de la segmentation de la ligne de log. Cinq signatures sont concernées dans le module de découpage et normalisation. En plus des champs extraits de la ligne de log, quatre autres champs sont dérivés à partir de l'URL. Ces nouveaux champs sont détaillés dans le Tableau 5.1. L'intérêt de ce découpage et de l'ajout de nouveaux champs dérivés réside dans le fait que *WebAnalyzer* peut ainsi appliquer ses signatures de façon spécifique et ciblée. Cette notion d'extraction ciblée de caractéristiques est présentée dans la Section 5.3.

Champ source	Champ destination	Transformation
URL	CURL	L'URL est nettoyée par la transformation d'un ensemble d'encodages dans des valeurs nominales. Par exemple, les caractères ASCII acceptés par le protocole HTTP sont substitués aux valeurs hexadécimales ou unicode qui peuvent être présentes dans l'URL. Les chemins redondants (par exemple /../toto/../) sont remplacés (l'exemple devient /../). Chacune de ces transformations est identifiée par un nom et ce nom est associé à la ligne de log si la transformation est effectuée.
CURL	HURL	L'URL nettoyée est transformée en substituant les caractères ASCII par leurs valeurs hexadécimales, pour permettre la recherche de code exécutable (code assembleur) dans les URL.
CURL	QUERY	L'URL nettoyée est découpée en deux parties, la ressource (script CGI) et les arguments. Ce champ contient la ressource, incluant le chemin d'accès. Ce chemin est relatif à l'arborescence du serveur HTTP. Par script CGI, nous entendons tout contenu actif (PHP, ASP, etc.)
CURL	ARGS	L'URL nettoyée est découpée en deux parties, la ressource et les arguments. Ce champ contient les arguments passés aux scripts CGI identifiés dans le champ QUERY. Dans le cas d'une requête pour une page statique, ce champ est vide.

TAB. 5.1 – Description des champs supplémentaires disponibles à l'issue du processus de normalisation

L'ajout de ces champs a pour objectif de faciliter l'écriture de signatures et de déjouer autant que possible certaines méthodes d'évasions. La présence des champs ajoutés facilite la recherche de certaines caractéristiques en présentant l'URL sous différentes formes.

Finalement, à partir d'une ligne de log, nous obtenons treize champs qui peuvent être individuellement inspectés lors de la phase d'extraction de caractéristiques.

5.2.2 Extraction de caractéristiques

Le module d'extraction de caractéristiques recherche non seulement des manifestations d'attaques mais aussi des caractéristiques descriptives. Les manifestations d'attaques sont le nom d'un script vulnérable ou la présence de code binaire destiné à une attaque par *buffer overflow* par exemple. Les caractéristiques descriptives peuvent être les extensions, et donc le type de ressource demandées, ou la présence d'arguments dans la requête. Cette collecte d'information a pour but de décrire de façon précise chaque ligne de log et, en cas d'attaque, de fournir à l'opérateur de sécurité une alerte contenant une description précise de l'attaque et des informations contextuelles. Ces informations sont utilisées dans le module de réconciliation et synthèse afin d'évaluer la dangerosité de l'attaque. L'extraction de caractéristiques se fait à partir d'un ensemble de signatures décrites dans la Section 5.3. Ces signatures utilisent des motifs

sous forme d'expressions régulières suivant la syntaxe Perl (*Perl Compatible Regular Expressions*). La recherche de motif est ciblée suivant les champs détaillés à la Section 5.2.1. Chaque signature est dotée d'un niveau de sévérité correspondant au niveau de danger que représente l'information découverte dans la ligne de log. À la sortie du module d'extraction de caractéristiques la ligne de log segmentée et enrichie est associée à un ensemble de signatures dont les motifs auront été trouvés et à un niveau de sévérité. Le niveau de sévérité correspond à la somme des sévérités des signatures qui auront été activées. L'extraction de caractéristiques utilise actuellement un jeu de 655 signatures.

Nous avons évoqué dans la Section 5.1 le coût de l'utilisation des algorithmes de *pattern matching* en détection d'intrusions. *WebAnalyzer* effectue une recherche aussi exhaustive que possible afin de récolter le plus d'information. Cette recherche se fait suivant un contrôle de flux permettant de sélectionner des signatures susceptibles d'être activées et d'éliminer au plus tôt celles dont on sait que dans un contexte donné, elle ne peuvent être activées. Cette notion de contrôle de flux ou de détection conditionnelle est exposée dans la Section 5.3.

5.2.3 Réconciliation et synthèse

Le module de réconciliation et synthèse utilise les informations portées par les signatures activées pendant la phase d'extraction de caractéristiques. Le raisonnement du module de réconciliation et synthèse se fait à partir de différents types d'information :

- la présence d'une vulnérabilité (CGI vulnérable, encodage spécifique, ...) ;
- l'objectif de l'attaque (exécution de commande, lecture de données, *scan*, ...) ;
- le succès de l'attaque (réussie, échouée, indéterminée).

Chacune de ces informations sert à moduler la sévérité finale attribuée à chaque ligne de log. Chaque signature porte un nom et appartient à une ou plusieurs classes. Le module de réconciliation et synthèse raisonne à partir de l'association de ces informations. Les associations sont décrites sous formes de signatures et exprimées en logique des prédicats d'ordre 1. Notre implémentation utilise l'intégration d'un moteur Prolog pour construire ce raisonnement. Le module de réconciliation et synthèse utilise 6 signatures. Ce nombre restreint est principalement dû au fait que les signatures de ce module ne raisonnent pas directement sur les signatures activées pendant la phase d'extraction de caractéristiques mais sur les classes auxquelles elles appartiennent.

Au sortir de la chaîne de traitement illustrée dans la Figure 5.3, l'opérateur dispose d'une ligne de log associée à un niveau de sévérité, des noms des signatures activées lors des traitements effectués par les trois modules, des classes et des sections auxquelles appartiennent ces signatures. L'ensemble de ces informations permettent d'évaluer la dangerosité d'une attaque et de la contextualiser afin de prendre les contre-mesures les mieux adaptées.

```
<signature name="php_ext" trigger="\.php$" severity="0" class="php">
  <description origin="vendor-specific">
    This indicates that the invoked cgi is a php script.
  </description>
</signature>
```

FIG. 5.4 – Exemple de signature utilisée par *WebAnalyzer*

5.3 Langage de signatures

Notre langage de signature est fondé sur un contrôle de flux concernant la recherche de signatures dans une ligne de log. Dans la Section 5.3.1 nous décrivons le format d'écriture des signatures utilisées par *WebAnalyzer*. Nous expliquons dans la Section 5.3.2 la mise en œuvre du contrôle de flux et de la détection conditionnelle grâce à la notion de `section`.

La langage de signatures que nous utilisons se base sur une syntaxe au format XML régie par un *Document Type Definition* (DTD). Le DTD définie tout d'abord les règles d'écritures des signatures utilisées par *WebAnalyzer*.

5.3.1 Structure de signature

La Figure 5.4 montre une signature telle qu'on en trouve dans le fichier de signatures. Cette signature a pour but de déterminer si la requête formulée par un client vise un script PHP sur les serveur web en recherchant l'extension « `.php` » dans le champ QUERY.

Une signature se compose de cinq champs : un nom (`name`), une fonction booléenne (`trigger`), une sévérité (`severity`), une liste de classes (`class`) et une description (`description`).

Le nom de la signature (`name`) est une clé identifiant la signature. Le nom n'est pas nécessairement unique, une même propriété pouvant être décrite par plusieurs signatures. Le nom de la signature servant au diagnostic, éviter les doublons permet de faciliter la compréhension du diagnostic par l'opérateur. Si la signature est activée, son nom est associée à la ligne de log.

La fonction booléenne (`trigger`) détermine si une signature est activée ou non. Cette fonction prend trois arguments : un champ de la ligne de log, la liste des noms des signatures déjà activées par la ligne de log et la liste des classes portées par ces signatures. L'implémentation actuelle de *WebAnalyzer* supporte deux types de fonctions booléennes : les expressions régulières et les règles Prolog. Le type de fonction contenu dans une signature est spécifié par la section à laquelle elle appartient.

La sévérité (`severity`) est une valeur numérique entière qui reflète le niveau de dangerosité de l'exploitation de la vulnérabilité que modélise la signature. Si la signature est activée, son niveau de sévérité est ajoutée à la sévérité de la ligne de log. Le choix de la sévérité portée par une signature est un travail d'expert. Dans notre implémentation, par exemple, la sévérité liée à l'exploitation d'un script CGI vulnérable est comprise entre 1 et 3, en fonction de l'ancienneté de la vulnérabilité et du danger que représente

l'exploitation de cette vulnérabilité. Dans l'exemple de la Figure 5.4, la signature indique à l'opérateur que la ressource demandée est un script PHP. Cette signature est informative et ne révèle pas de danger particulier d'où sa sévérité nulle.

La notion de classe (`class`) caractérise des propriétés communes à plusieurs signatures. Suivant l'exemple présenté à la Figure 5.4, toutes les signatures permettant de déterminer que la ressource demandée par un client est un script PHP portent la classe `php`. Une signature peut porter plusieurs classes en fonction des caractéristiques qu'elle décrit. Par exemple, la signature qui reconnaît une tentative d'accès au fichier `/etc/shadow` portent les classes `unix` et `file`. La classe `unix` indique que le système d'exploitation visé par l'attaquant est de type Unix. La classe `file` indique que l'attaquant cherche à accéder à un fichier sensible du système (ici, le fichier de mots de passe du serveur).

L'expression de ces propriétés nous permet donc de reconnaître rapidement le type de serveur cible d'une attaque (par exemple IIS ou Apache), le système d'exploitation (Windows, Unix) et l'application visée (par exemple frontpage, coldfusion). Elle permet aussi de qualifier la technique d'attaque utilisée : *buffer overflow*, *directory traversal*, *cross-site scripting*, injection de code SQL, encodage malicieux ou abus des fonctions du protocole HTTP. Finalement ces classes permettent de déterminer l'objectif de l'attaque : exécution de code, accès à des fichiers sensibles, visualisation de contenu de répertoires, déni de service ou installation de cheval de troie. Dans le cas où une signature est activée, les classes qu'elle porte sont associées de manière non redondante à la ligne de log. Le fichier de signatures dispose actuellement de 59 classes. Les classes sont utilisées par les signatures Prolog dans le module de réconciliation et synthèse afin de déterminer la sévérité finale à associer à une requête.

La description (`description`) est une structure XML reprenant les informations demandées dans le champ *Classification* du modèle IDMEF [88] documentant la signature. Ces informations contiennent en particulier des des références de type *Common Vulnerabilities and Exposures* (CVE).

Signatures à base d'expressions régulières

Ces signatures sont utilisées uniquement dans le module d'extraction de caractéristiques. Les signatures à base d'expressions régulières utilisent le *pattern matching* afin de découvrir des caractéristiques spécifiques dans les différents champs dérivés d'une ligne de log. Comme nous l'avons vu avec l'exemple présenté à la Figure 5.4, ces signatures peuvent servir à extraire des caractéristiques simples permettant de décrire en détail une ligne de log.

L'utilisation des expressions régulières nous permet d'écrire facilement des signatures complexes. La Figure 5.5 montre la signature utilisée pour détecter une requête vers un shell Unix, quelque soit le type de shell demandé. Cette signature détecte aussi bien une tentative d'accès à un shell `bash` qu'à un shell `ksh`. La souplesse des expressions régulières nous permet d'avoir à maintenir qu'une seule signature pour toute tentative d'accès à un shell Unix. De façon générale, cela permet de n'avoir qu'une seule signature pour détecter des attaques ayant des caractéristiques similaires, c'est-à-dire des motifs « proches », une même dangerosité et dénotant les mêmes classes. Le fait de

```
<signature name="unix_shells"
           trigger="/(a|ba|cs|z|tc|r|rk|k)?sh$"
           severity="1"
           class="query,cgi,shell">
  <description origin="cve">
    <name>CAN-1999-0509</name>
    <url>http://</url>
  </description>
</signature>
```

FIG. 5.5 – Signature de recherche de shells

```
  <signature name="success_file"
        trigger="pattern(status_200),class(file)"
        severity="+3"        class="rule">
    <description origin="vendor-specific">
      <name>If a FILE referenced as dangerous has an OK status, then
      the severity is increased.</name>
      <url>file://./signatures.xml</url>
    </description>
  </signature>

  <signature name="failed_file"
        trigger="pattern(notfound_404),class(file)"
        severity="-1"        class="rule">
    <description origin="vendor-specific">
      <name>If a FILE referenced as dangerous has an explicit failed
      status, then the severity is decreased.</name>
      <url>file://./signatures.xml</url>
    </description>
  </signature>
```

FIG. 5.6 – Réconciliation entre fichier et code retour

pouvoir concentrer plusieurs manifestations d'attaques dans une même signature simplifie la maintenance du fichier de signatures, celles-ci étant moins nombreuses. Cette simplification est à mettre en rapport avec la complexité du motif contenu dans ces signatures.

Les performances en terme de recherche sont, dans ce cas, fonction de la bibliothèque utilisée pour mettre en œuvre le *pattern matching*.

Signatures à base de règles Prolog

Ces signatures sont uniquement utilisées dans le module réconciliation et synthèse. Les signatures à base de règles Prolog utilisent les informations portées par la ligne de log à la fin du traitement effectué par le module d'extraction de caractéristiques. L'exploitation des noms de signatures activées et des noms de classes associés à une ligne de log permet de moduler la sévérité qui lui est finalement attribuée.

La Figure 5.6 montre deux signatures utilisant des règles Prolog. La première signature reconnaît un accès réussi à un fichier sensible. Le prédicat `pattern(status_200)` reconnaît la présence de la signature nommée `status_200` (le serveur a répondu po-

```
<section type="pattern" name="phpEx" field="query"
         streamControl="jump" prerequisite="()">
  [SIGNATURES]
</section>
<section type="pattern" name="knowncgis_phpEx" field="query"
         streamControl="jump" prerequisite="(phpEx)">
  [SIGNATURES]
</section>
```

FIG. 5.7 – Exemple de sections permettant une détection conditionnelle

sitivement à la requête du client) associée à la ligne de log. Le prédicat `class(file)` reconnaît la présence de la classe `file` (tentative d'accès à un fichier sensible) associée à la ligne de log. La seconde signature reconnaît elle une tentative échouée (`pattern(status_400)`) d'accès à un fichier sensible (`class(file)`). On observe que les signatures Prolog peuvent utiliser les noms de signatures activées (prédicat `pattern()`) et les classes portée par ces signatures (prédicat `class()`). Les signatures basées sur des règles Prolog sont utilisées pour caractériser des tentatives d'accès à des scripts potentiellement vulnérables ou à des fichiers systèmes, utilisant le code retour du serveur (*status code*) pour déterminer si l'attaquant a réussi son attaque ou pas. En plus de ces règles génériques, nous identifions précisément certaines attaques, souvent difficiles à caractériser par une unique signature basée sur des expressions régulières.

5.3.2 Organisation des signatures et procédé d'évaluation

Les signatures utilisées pour l'analyse permettent d'avoir une description complète des requêtes effectuées par les clients. Notre objectif reste cependant d'effectuer cette analyse en un minimum de temps. Nous organisons les signatures afin d'éviter les évaluations inutiles.

Les signatures utilisées par *WebAnalyzer* sont regroupées sous forme de sections, selon les propriétés communes qu'elles partagent. Ces propriétés communes concernent l'information portée par les signatures (par exemple les signatures recherchant l'exploitation d'une vulnérabilité dans des scripts PHP), le type de fonction booléenne utilisée (*pattern matching*, règle Prolog) et le champ de la ligne de log sur lequel s'effectue la détection.

La Figure 5.7 montre un extrait tronqué de notre fichier de signature où l'on voit deux structures de sections. Une structure de section se compose de cinq champs : un type de section (`type`), un nom (`name`), le champ visé (`field`), un contrôle de flux (`streamControl`) et un ensemble de pré-requis (`prerequisite`).

Le type de section défini la fonction booléenne utilisée par les signatures qui composent la section. Une section de type `pattern` implique que les signatures utilisent un mécanisme d'expressions régulières et une section de type `rule` implique que les signatures utilisent des règles Prolog.

Le nom de la section est une clé identifiant la section. Ce nom n'est pas nécessairement unique, mais il identifie le type de signatures contenu dans la section.

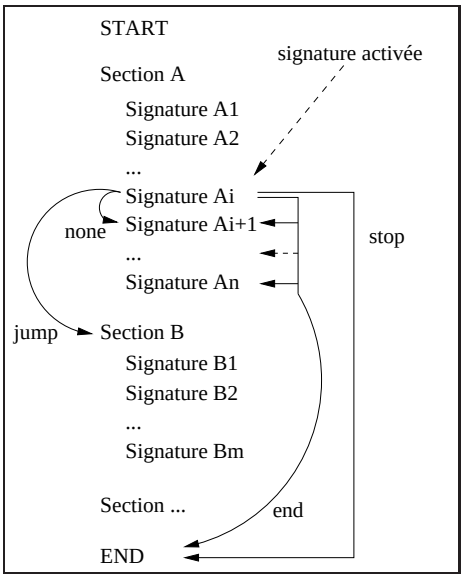

FIG. 5.8 – TITRE

Le champ visé identifie un champ issu de la segmentation de la ligne de log. C'est sur ce champ que la fonction booléenne contenu dans les signatures appartenant à la section s'applique.

Le champ `streamControl` définit des actions à entreprendre lorsqu'une signature de la section est activée. Ce champ peut prendre les valeurs suivante : `stop`, `end`, `jump`, `none`. La Figure 5.8 illustre le traitement effectué en fonction de la valeur du champ `streamControl` lorsqu'une signature est activée. La valeur `stop` indique que le processus de détection s'arrête directement et que l'événement et les signatures qui lui sont associées sont renvoyées à l'opérateur. La valeur `end` indique que le processus de détection doit continuer jusqu'à la fin de la section et s'arrêter. La valeur `jump` indique que le processus de détection doit se poursuivre à partir de la section suivante. la valeur `none` indique qu'il n'y a aucune action à effectuer. Suivant l'exemple présenté à la Figure 5.7, on observe que le champ `streamControl` a pour valeur `jump` dans la section `knowncgis_phpEx`. Si une des signatures de cette section est activée, le processus de détection continue directement à la section suivante, sans tester les signatures restantes dans la section. Ce premier contrôle de flux permet d'exclure un ensemble de signatures du processus de détection au sein d'une section. Il impose aussi que les signatures contenues dans une section où le champ `streamControl` a pour valeur `jump` soient exclusives les unes des autres ou au moins ordonnées, de sorte que des signatures qui peuvent être pertinentes ne soient pas exclues du processus de détection.

L'ensemble des pré-requis est une expression booléenne associant des noms de signatures, des noms de sections et des noms de classes. Si cette expression booléenne est évaluée à « vrai », c'est-à-dire que la ligne de log en cours de traitement est associée à des noms de signatures, de sections et de classes correspondant à tous ceux présents dans

l'expression booléenne, alors le processus de détection commence à parcourir les signatures présentes dans la section. L'exemple de la Figure 5.7 présente une première section contenant des signatures dédiées à la reconnaissance de script PHP. La seconde section contient des signatures recherchant l'exploitation de vulnérabilité dans des scripts PHP. Si la ligne de log active une signature appartenant à la section `phpEx`, le nom de la section lui est associé. L'application des signatures de la section `kowncgis_phpEx` est conditionnée par la présence du nom de section `phpEx` associé à la ligne de log en cours de traitement. Ce mécanisme nous permet d'exclure des sections entières du processus de détection et ainsi d'éviter des analyses inutiles. Nous définissons une section pour chaque type de script (Perl, PHP, ASP, ...) qui permet ensuite de cibler l'analyse sur les signatures relatives à ce type de script. Dans l'exemple de la Figure 5.7, le test d'absence de script PHP (testé grâce à quatre signatures) permet d'exclure les 41 signatures relatives aux vulnérabilités de script PHP. De même, nous pouvons exclure la recherche de données particulières dans les arguments de la ligne de log si le champ ARGS est vide.

Le mécanisme de détection conditionnelle au niveau des signatures et des sections nous permet de construire des arbres de décision, ce qui optimise le processus de détection d'intrusions. Cette construction se fait manuellement lors de la création des signatures et de leur ordonnancement dans les sections. L'exclusivité « naturelle » du type de données traitées (un script ne peut pas être à la fois un script PHP et un script Perl par exemple) nous assure que la détection conditionnelle est efficace au niveau des sections et occasionne un sur-coût largement compensé par l'exclusion d'un grand nombre de signature du processus de détection.

Cependant, l'écriture et l'organisation des signatures au sein d'une même section ne sont pas triviales et demandent un effort à l'opérateur de sécurité. Une seule signature pouvant interrompre le processus de détection, il est nécessaire d'apporter un soin particulier à la qualité du motif qui est recherché et à la position de la signature dans la section.

Le mécanisme de détection conditionnelle peut également être utilisé pour une analyse de type comportementale. Il est possible d'écrire des signatures dédiées à la reconnaissance de ressources d'un serveur donné et ainsi d'éviter l'analyser de requêtes vers des ressources connues et non critiques. Ces ressources sont reconnues au début du processus de détection et l'analyse s'arrête. Si le nombre de ressources est trop important, on risque de surcharger le fichier de signatures, un site web comptant facilement plusieurs milliers de ressources. Le recours à des motifs reconnaissant partiellement des ressources ou groupes de ressources connues et non-critiques (reconnaissance des extensions de fichier HTML par exemple) peut être utilisé. Cependant, on s'expose dans ce cas à l'exploitation d'attaques par contournement (*mimicry attacks*) trompant ces signatures « partielles ». Finalement, la mise en œuvre et la maintenance d'un tel mode d'utilisation de *WebAnalyzer* peut être extrêmement coûteuse en temps pour un opérateur de sécurité.

Nom	Type	Sévérité	Description
`notfound_404`	pattern	0	Identifie le code de retour émis par le serveur. La ressource pointée par l'URL n'a pas été trouvée sur le serveur.
`status_200`	pattern	0	Identifie le code de retour émis par le serveur. La requête a été normalement satisfaite.
`not_allowed_40x`	pattern	+1	Identifie le code de retour émis par le serveur. Nous assimilons les codes `400` à `403` à des actions n'ayant pas réussi, mais pour lesquelles le serveur a pu avoir une interaction avec l'attaquant.
`get_method`	pattern	0	Identifie la méthode de la requête.
`non_ascii`	pattern	+1	Caractérise la présence de caractère(s) non imprimable(s) dans l'URL (ici `%0a`).
`cgi_dir`	pattern	0	Identifie le fait qu'un répertoire contenant potentiellement des scripts CGI (/cgi-bin/) a été trouvé dans l'URL.
`phf`	pattern	+1	Caractérise la vulnérabilité `CA-1996-06`.
`etc_password`	pattern	+1	Caractérise la cible `/etc/passwd`.
`unix_cmd`	pattern	+1	Caractérise la présence dans l'URL d'une commande système unix (ici, `/bin/cat`).
`args_not_empty`	pattern	0	Indique la présence d'arguments.
`real_attempt`	rule	+1	Raisonne sur les signatures `phf` et `args_not_empty`. Caractérise l'accès à un script vulnérable combiné à l'utilisation de paramètres.
`failed_file`	rule	-1	Raisonne sur les signatures `etc_password` et `notfound_404`. L'attaquant n'a pas pu accéder à un fichier système.
`failed_cgi`	rule	-1	Raisonne sur les signatures `phf` et `notfound_404`. L'attaquant n'a pas pu accéder à un script vulnérable.
`success_cgi`	rule	+3	Raisonne sur les signatures `phf` et `status_200`. L'attaquant a eu accès à un script vulnérable.
`success_file`	rule	+3	Raisonne sur les signatures `etc_passwd` et `status_200`. L'attaquant a eu accès à un fichier système.

TAB. 5.2 – Description des signatures utilisées dans les exemples de diagnostic de *WebAnalyzer*

5.4 Exemples de fonctionnement du WebAnalyzer

Dans cette section, nous illustrons les capacités de diagnostic de *WebAnalyzer* en analysant des lignes de log que nous avons forgées. Ces lignes de log sont une déclinaison possible d'une requête vers un script présentant une vulnérabilité. Chacune de ces déclinaisons est présentée dans différents contextes de réponse du serveur web

La vulnérabilité `CA-1996-06` concerne le script CGI `phf`. L'utilisation de certains caractères d'échappement (ici `%0a`) permet de faire exécuter des commandes sur le serveur via le script `phf`. La première requête consiste en une simple tentative d'accès, sans argument, tel que le ferait un attaquant pour vérifier la présence du script vulnérable. La seconde requête utilise un fichier de mot de passe comme argument dans la requête, sans utiliser pour autant la vulnérabilité propre au script `phf`. La dernière requête est une attaque réelle vers le script `phf`, utilisant la vulnérabilité `CA-1996-06`. Les différents

contextes sont les codes de retour émis par le serveur web. Les codes de retour utilisés sont `200`, `403` et `404`. Nous obtenons ainsi neuf lignes de log à analyser.

Les résultats obtenus sont illustrés dans les Figures 5.9, 5.10 et 5.11. Le résultat de l'analyse de chaque ligne de log se présente sur cinq lignes. La première ligne est la ligne de log telle qu'elle est lue par *WebAnalyzer*. La deuxième ligne présente la sévérité attribuée par *WebAnalyzer* à la ligne de log. Les troisième, quatrième et cinquième lignes sont respectivement les listes de signatures, sections et classes qui sont activées par la ligne de log lors de son analyse.

Le tableau 5.2 décrit les signatures impliqués dans les résultats d'analyse que nous présentons dans les Sections 5.4.1, 5.4.2 et 5.4.3.

Les trois premières signatures concernent le code de retour émis par le serveur web. Ces signatures ne concernent pas à jour les attaques subies par le serveur web, mais permettent éventuellement d'associer un contexte à une attaque. Elle permettent par exemple de déterminer si une attaque a réussi ou non.

Les quatrième signature concerne la méthode HTTP utilisée par le client (ici `GET`). Les méthodes `GET` et `POST` sont des méthode habituellement utilisées et sont associées à une sévérité nulle. En revanche, des méthodes telles que `HEAD` ou `OPTIONS` sont plus marginales et plus souvent utilisées dans des contextes de *scans* ou d'attaques. Comme pour le code de retour émis par le serveur, la méthode HTTP n'est pas une preuve d'attaque mais permet *a posteriori*, lors de la phase de réconciliation, de contextualiser une éventuelle attaque et donc d'en ré-évaluer le niveau de sévérité.

Les six signatures suivantes identifient respectivement l'utilisation d'encodage suspects, un accès vers le répertoire où se trouve les scripts CGI, un accès vers un script dont une vulnérabilité est connue, la présence d'un nom de fichier système, la présence d'une commande Unix et la présence de paramètres.

Les cinq dernières signatures sont des règles Prolog permettant de ré-évaluer la sévérité associée à la ligne de log en fonction des signatures précédemment activées.

5.4.1 Tentative d'accès

Les tentatives d'accès sont illustrées à la Figure 5.9. La première requête est une tentative d'accès échouée (code de retour `404`). Le niveau de sévérité nul est dû à la signature `failed_cgi`. Le niveau de sévérité 0 indique qu'aucune alerte n'est émise. En effet, nous considérons que les *scans* font partie du « bruit de fond » sur Internet et dans ce cas, la réponse négative du serveur montre qu'il n'est pas vulnérable.

La seconde requête présente une tentative d'accès réussie vers le script `phf`, n'exploitant pas la vulnérabilité `CA-1996-06`. Le niveau de sévérité 4, dû à la signature `success_cgi`, est assez faible et indique que le serveur web n'a pas été compromis. Cependant, la découverte de la présence de ce script vulnérable sur le serveur doit amener une contre-mesure ou une vérification de la part de l'opérateur de sécurité. En effet, une fois la présence du script établie par un attaquant, ce dernier peut exploiter la vulnérabilité propre à ce script.

Le code de retour de la troisième requête ne permet pas de déterminer avec certitude la portée de celle-ci. En effet, *WebAnalyzer* ne peut déterminer si la requête a réussi

```
Processing: 1.1.1.1 - - [26/Feb/2002:18:37:19 -0500] "GET /cgi-bin/phf HTTP/1.0" 404 310
Severity  : 0
Alerts    : notfound_404,get_method,cgi_dir,phf,failed_cgi
Classes   : status,method,query,apache,cgi,rule
Sections  : status,method,cgiId,knowncgis_cgiId,priorite
-------------
Processing: 1.1.1.2 - - [26/Feb/2002:18:37:19 -0500] "GET /cgi-bin/phf HTTP/1.0" 200 310
Severity  : 4
Alerts    : status_200,get_method,cgi_dir,phf,success_cgi
Classes   : status,method,query,apache,cgi,rule
Sections  : status,method,cgiId,knowncgis_cgiId,priorite
-------------
Processing: 1.1.1.3 - - [26/Feb/2002:18:37:19 -0500] "GET /cgi-bin/phf HTTP/1.0" 403 310
Severity  : 2
Alerts    : not_allowed_40x,get_method,cgi_dir,phf
Classes   : status,method,query,apache,cgi
Sections  : status,method,cgiId,knowncgis_cgiId
```

FIG. 5.9 – Exemple de résultats d'analyse de *WebAnalyzer* relatifs à des tentatives d'accès au script `phf`

ou a échoué. Le serveur a aussi bien pu révéler de l'information concernant la présence du script `phf` sur le serveur que connaître un problème interne. Aucune des deux signatures précédentes (`failed_cgi` et `success_cgi`) ne s'applique dans ce cas et aucune ré-évaluation de la sévarité n'est effectuée.

5.4.2 Passage d'argument illicite

L'analyse des trois requêtes présentées à la Figure 5.10 sont relatives à une tentative d'accès à un fichier système via l'utilisation d'un script vulnérable. Cependant, la vulnérabilité propre au script `phf` n'est pas exploitée.

En fonction de la réponse apportée par le serveur web, les sévérités associées à chacune des requête vont de 2 à 9. La première requête consiste en une tentative d'accès échouée. Comparé à la simple tentative d'accès échouées vue précédemment, en plus de la signature `failed_cgi`, nous avons les signatures `etc_passwd` et `failed_file` qui viennent modifier la sévérité au niveau 2.

Pour la seconde requête, l'activation des signatures `fsuccess_cgi` et `success_file` constitue un facteur de dangerosité aggravant, positionnant le niveau de sévérité à 9. En effet, dans ce cas, l'accès à un scripte présentant une vulérabilité (non exploitée ici) a permis à un attaquant d'accéder à des données sensibles du système.

Concernant la troisième requête, le code de retour `403` ne permet pas de déterminer la portée de l'attaque. Cependant, la présence d'un nom de fichier système dans la requête et la tentative d'accès à un script vulnérable positionne la sévérité au niveau 4, attirant l'attention de l'opérateur de sécurité.

5.4.3 Exploitation de la vulnérabilité

Le résultat d'analyse présenté à la Figure 5.11 concerne l'exploitation de la vulnérabilité du script `phf` visant à récupérer le contenu du fichier `/etc/passwd`. Il s'agit

```
Processing: 1.1.1.4 - - [26/Feb/2002:18:37:19 -0500]
            "GET /cgi-bin/phf?/etc/passwd HTTP/1.0" 404 310
Severity  : 2
Alerts    : notfound_404,get_method,cgi_dir,phf,etc_password,
            args_not_empty,real_attempt,failed_file
Classes   : status,method,query,apache,cgi,file,unix,args,rule
Sections  : status,method,cgiId,knowncgis_cgiId,genericCurl,argsExists,priorite
-------------
Processing: 1.1.1.5 - - [26/Feb/2002:18:37:19 -0500]
            "GET /cgi-bin/phf?/etc/passwd HTTP/1.0" 200 310
Severity  : 9
Alerts    : status_200,get_method,cgi_dir,phf,etc_password,
            args_not_empty,real_attempt,success_cgi,success_file
Classes   : status,method,query,apache,cgi,file,unix,args,rule
Sections  : status,method,cgiId,knowncgis_cgiId,genericCurl,argsExists,priorite
-------------
Processing: 1.1.1.6 - - [26/Feb/2002:18:37:19 -0500]
            "GET /cgi-bin/phf?/etc/passwd HTTP/1.0" 403 310
Severity  : 4
Alerts    : not_allowed_40x,get_method,cgi_dir,phf,etc_password,
            args_not_empty,real_attempt
Classes   : status,method,query,apache,cgi,file,unix,args,rule
Sections  : status,method,cgiId,knowncgis_cgiId,genericCurl,
            argsExists,priorite
```

FIG. 5.10 – Exemple de résultats d'analyse de *WebAnalyzer* relatifs au passage d'argument illicite au script `phf`

là de trois attaques correctement formulées, telles que décrites par le bulletin d'alerte `CA-1996-06`.

La première requête est une attaque echouée. La sévérité est ramenée au niveau 4 par les signatures `failed_file` et `failed_cgi`. Là encore, le niveau de sévérité 4 est censé attirer l'attention de l'opérateur de sécurité, mais indique que le serveur n'a pas été compromis par l'attaque.

En revanche, pour la seconde requête, le niveau de sévérité 11, dû aux signatures `success_file` et `success_cgi` révèle la dangerosité de l'attaque. Ici, l'attaquant a réussi, grâce à l'exploitation de la vulnérabilité `CA-1996-06`, à accéder directement au fichier `/etc/passwd`. Dans ce cas, le niveau de sévérité élevé permet à l'opérateur de sélectionner directement cette alerte et de prendre les contre-mesures nécessaires au plus tôt.

Le code de retour de la troisième requête ne permet pas de ré-évaluer sa sévérité. Cependant l'activation cumulée des signatures `phf`, `non_ascii`, `unix_cmd`, `etc_passwd` et `real_attempt` porte la sévérité liée à cette requête au niveau 6 et la détache complètement des requêtes dont on sait qu'elles n'ont pas pu compromettre le serveur. Pour cette requête, l'opérateur de sécurité devra enquêter plus avant pour déterminer la portée de l'attaque.

5.5 Conclusion

L'approche que nous avons présentée avec *WebAnalyzer* apporte un diagnostic très précis à à l'opérateur de sécurité. La recherche de vulnérabilités, la recherche de ca-

```
Processing: 1.1.1.7 - - [26/Feb/2002:18:37:19 -0500]
            "GET /cgi-bin/phf?Qalias=%0a/bin/cat%20/etc/passwd HTTP/1.0" 404 310
Severity  : 4
Alerts    : non_ascii,notfound_404,get_method,cgi_dir,phf,etc_password,
            unix_cmd,args_not_empty,real_attempt,failed_cgi,failed_file
Classes   : non_ascii,status,method,query,apache,cgi,file,unix,exec,
            linux,args,rule
Sections  : miscellaneous,status,method,cgiId,knowncgis_cgiId,
            genericCurl,argsExists,priorite
-------------
Processing: 1.1.1.8 - - [26/Feb/2002:18:37:19 -0500]
            "GET /cgi-bin/phf?Qalias=%0a/bin/cat%20/etc/passwd HTTP/1.0" 200 310
Severity  : 11
Alerts    : non_ascii,status_200,get_method,cgi_dir,phf,etc_password,
            unix_cmd,args_not_empty,real_attempt,success_cgi,success_file
Classes   : non_ascii,status,method,query,apache,cgi,file,unix,exec,
            linux,args,rule
Sections  : miscellaneous,status,method,cgiId,knowncgis_cgiId,
            genericCurl,argsExists,priorite
-------------
Processing: 1.1.1.9 - - [26/Feb/2002:18:37:19 -0500]
            "GET /cgi-bin/phf?Qalias=%0a/bin/cat%20/etc/passwd HTTP/1.0" 403 310
Severity  : 6
Alerts    : non_ascii,not_allowed_40x,get_method,cgi_dir,phf,
            etc_password,unix_cmd,args_not_empty,real_attempt
Classes   : non_ascii,status,method,query,apache,cgi,file,unix,exec,
            linux,args,rule
Sections  : miscellaneous,status,method,cgiId,knowncgis_cgiId,
            genericCurl,argsExists,priorite
```

FIG. 5.11 – Exemple de résultats d'analyse de *WebAnalyzer* relatifs à l'exploitation de la vulnérabilité du script `phf`

ractéristiques critiques habituellement utilisées dans des attaques, ainsi que la contextualisation des requêtes permet d'associer des niveaux de sévérité aux attaques en fonction de leur portée.

L'exemple d'analyse que nous avons présenté illustre la hiérarchisation du diagnostic émis par *WebAnalyzer*. En ordonnant ainsi les alertes suivant leur niveau de sévérité, on permet à l'opérateur de sécurité de s'occuper en premier lieu des alertes les plus dangereuses pour le serveur avant de se pencher sur des événements plus bénins.

De plus, le langage de signatures mis en œuvre permet d'organiser une détection optimale, éliminant de l'analyse les signatures inopportunes. L'écriture de nouvelles signatures, telles qu'organisées dans notre format XML est à la portée de tout opérateur de sécurité.

Une caractéristique intéressante de notre approche est la recherche d'éléments propres à participer à une attaque (tels que les encodages non ASCII), sans pour autant être liés à une vulnérabilité connue. Cette caractéristique peut permettre de découvrir de nouvelles attaques et donc de limiter les faux négatifs.

Mais cette caractéristique est à double tranchant. En effet, elle rend *WebAnalyzer* extrêmement sensible. *WebAnalyzer* peut ainsi générer facilement des alertes qui n'ont pas lieu d'être. Ce problème est perceptible dans les résultats expérimentaux présentés au Chapitre 6. Cependant, une adaptation du fichier de signatures au type de serveur

surveillé, ainsi qu'à l'organisation des données qu'il héberge peut permettre de limiter l'émission de fausses alertes.

Finalement, une des limitation évidente de notre approche est que l'utilisation de fichiers de log comme source de données ne nous donne accès qu'à une petite part de la requête HTTP. Toute attaque entièrement mise en œuvre dans le corps de la requête HTTP ne saurait être détectée par *WebAnalyzer*. Pour pallier ce problème, *WebAnalyzer*pourrait être positionnée comme un module du serveur web, analysant l'intégralité de la requête HTTP soumise au serveur web.

Chapitre 6

Résultats expérimentaux

Dans ce chapitre nous présentons tout d'abord les résultats expérimentaux que nous avons obtenus avec les deux modules de détection d'intrusions que nous avons présentés au Chapitre 4 et 5, *WebFilter* et *WebAnalyzer*. Nous présentons ensuite les résultats de l'utilisation de ces deux modules suivant la combinaison en série $A_u \rightarrow M$ présentée dans le Chapitre 3.

Chacun des résultats que nous présentons dans ce chapitre sont relatifs à des données issus des serveurs web de Supélec, de l'IRISA et d'un serveur web interne à France Télécom. La création de modèles de comportements pour *WebFilter* des serveurs de l'IRISA et de France Télécom est présentée dans la Section 6.1 (le modèle de comportement du serveur de Supélec est celui établi au Chapitre 4). Les résultats d'analyses effectuées avec *WebAnalyzer* seul sur les trois serveurs web sont ensuite présentés dans la Section 6.2. Finalement nous présentons dans la Section 6.3 les résultats de la mise en œuvre de la combinaison en série $A_u \rightarrow M$ et comparons les capacités effectives d'une telle combinaison par rapport à l'utilisation de *WebAnalyzer* seul.

6.1 Le modèle de comportement de *WebFilter*

Nous avons décrit au Chapitre 4 le modèle de comportement du serveur web de Supélec. Dans cette section nous décrivons la création de modèles de comportement de *WebFilter* pour deux autres serveurs web. Le premier est le serveur de l'IRISA, serveur web académique. Le second est un serveur web interne au système d'information de France Télécom.

Dans le Chapitre 4, nous avons étudié les groupes formés à partir des données du serveur web de Supélec suivant la valeur et la variance des différentes variables. Cette étude nous a permis de dresser différents profils de groupe. Dans cette section, nous utilisons ces profils comme éléments de référence pour la sélection des groupes formés à partir des données des serveurs web de l'IRISA et de France Télécom. Le cas échéant, nous approfondissons l'interprétation des groupes formés. En effet, les serveurs web ayant *a priori* un public différent, il est possible d'observer des différences de comportement dans l'utilisation des ressources de ces serveurs. L'utilisation des profils établis

au Chapitre 4 permet une certaine automatisation dans la construction des modèles de comportement. Elle permet aussi de réduire l'expertise nécessaire à un opérateur de sécurité, ainsi que le temps investi dans la sélection des ressources constituant le modèle de comportement.

Après avoir sélectionné les ressources à utiliser dans le modèle de comportement, nous analysons le modèle de comportement créé suivant deux axes. Le premier axe concerne la complétude et la correction du modèle. Le second axe concerne les capacités de filtrage escomptées à partir du modèle de comportement créé. Ces capacités de filtrage seront effectivement mesurées dans la Section 6.3.

6.1.1 Le serveur web de l'IRISA

Nous utilisons les 11 derniers jours de log du mois de septembre 2004 du serveur web de l'IRISA (`http://www.irisa.fr`). Durant cette période, le serveur web a reçu 973 518 requêtes vers 67 626 ressources (soit environ 14 requêtes par ressources et 88 501 requêtes par jour en moyenne). On observe que le nombre de ressources demandées sur le serveur web de l'IRISA est plus de trois fois supérieur à celui du serveur web de Supélec (21 678 ressources). De même, on remarque que pour une durée d'observation trois fois moindre à celle accordée au serveur de Supélec, on a un nombre de requêtes équivalent.

Le grand nombre de ressources nous pose problème en pratique. En effet, les algorithmes et les mises en œuvre que nous utilisons ne semblent pas dimensionnés pour traiter de tels volumes. Si nous voulons traiter l'ensemble des ressources de l'IRISA (67 626 ressources), il nous faut faire d'importantes concessions sur le volume des échantillons utilisés pour déterminer le nombre de groupes à créer. Les résultats que nous obtenons alors ne nous permettent pas de choisir un nombre de groupes avec assez de certitude. De plus, les groupes que nous obtenons sont inexploitables. En effet, aucun profil connu ne se dégage pour les groupes formés. Les variances associées aux variables décrivant chaque groupe sont telles que nous ne pouvons pas utiliser les valeurs des variables pour interpréter correctement les regroupements et déterminer de nouveaux profils.

Nous avons choisi de répondre à ce problème en scindant notre corpus de ressources en trois sous-ensembles. Nous obtenons ainsi trois corpus de tailles équivalentes au corpus utilisé pour le serveur web de Supélec. La scission se fait suivant le premier niveau d'arborescence des ressources. Par exemple, on regroupe les ressources dont le chemin commence par `/accueil/` et celles commençant par `/biblio/` pour constituer le premier sous-ensemble, afin d'obtenir un sous-ensemble contenant un nombre équivalent de ressources à celui du serveur web de Supélec (environ 20 000 ressources). Nous nous autorisons ce découpage suivant le premier niveau de l'arborescence car ils regroupent des types de ressources similaires quand à l'utilisation qui en est faite par les clients, indépendamment les uns des autres. De ce point de vue, nous pouvons alors considérer les trois groupes de ressources que nous avons ainsi formé comme trois serveurs web différents. Suivant le découpage effectué, nous obtenons trois sous-ensembles de ressources constitués respectivement de 22 850 ressources, 22 174 ressources et 22 602

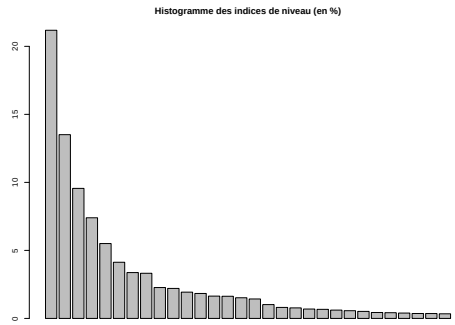

FIG. 6.1 – Histogramme des indices de niveaux pour une CAH réalisées sur un échantillon de 25% du premier sous-ensemble de données du serveur web de l'IRISA

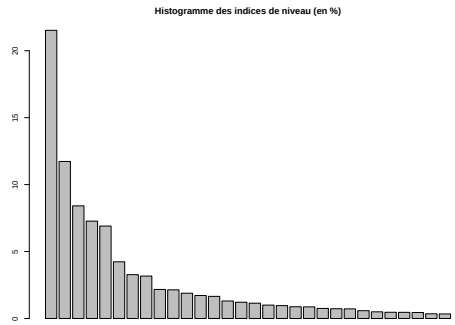

FIG. 6.2 – Histogramme des indices de niveaux pour une CAH réalisées sur un échantillon de 25% du deuxième sous-ensemble de données du serveur web de l'IRISA

ressources. Notons que les données statistiques associées à chaque individu sont calculées par rapport à l'ensemble des individus et non par rapport aux seuls individus du sous-ensemble auquel il appartient. En faisant ce choix, nous conservons, après le partitionnement des trois sous-ensembles, la possibilité de comparer entre eux des groupes issus de sous-ensembles différents. Par la suite, nous désignerons par 1.x les groupes issus du premier sous-ensemble des données de l'IRISA, par 2.x les groupes issus du deuxième sous-ensemble et par 3.x les groupes issus du troisième.

6.1.1.1 Choix du nombre de groupes à former

Suivant le mode opératoire établi au Chapitre 4, nous déterminons tout d'abord le nombre de groupes à former en effectuant des classifications ascendantes hiérarchiques (CAH) sur un échantillon de 25 % de chacun des trois sous-ensembles des données de

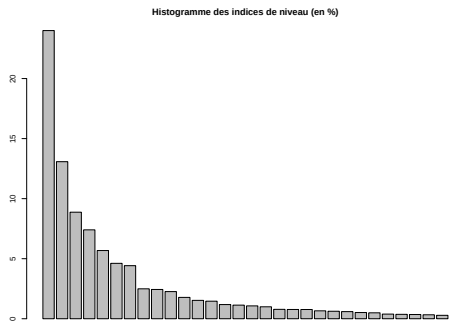

FIG. 6.3 – Histogramme des indices de niveaux pour une CAH réalisées sur un échantillon de 25% du troisième sous-ensemble de données du serveur web de l'IRISA

l'IRISA. Les Figures 6.1, 6.2 et 6.3 montrent les courbes d'indices de niveaux associées à des CAH effectuées sur des échantillons issus des trois sous-ensembles.

Suivant les courbes d'indices de niveaux, nous choisissons de partitionner le premier sous-ensemble en 9 groupes. En effet, on observe sur la Figure 6.1 que le coût d'agrégation jusqu'à 9 groupes est relativement faible. On peut remarquer aussi que le passage de 9 à 8 groupes est lui aussi peu coûteux, ce qui aurait pu nous conduire à étudier un partitionnement en 7 groupes du premier sous-ensemble. Cependant, ce choix aurait pu nous masquer une différence significative entre les deux groupes précédemment agrégés. De plus, choisir un nombre de groupes plus grand pour le partitionnement ne gêne *a priori* pas l'interprétation du partitionnement effectué. Nous risquons tout au plus d'obtenir quelques groupes très similaires après la phase de *clustering*. Notre interprétation de la deuxième courbe d'indice de niveau (Figure 6.2) nous conduit aussi à choisir un partitionnement en 9 groupes. Comme pour le premier sous-ensemble, on pourrait considérer un partitionnement en 6 ou 7 groupes. Cependant, là aussi nous avons choisi de conserver un plus grand nombre de groupes. Finalement, nous choisissons de partitionner le troisième sous-ensemble en huit groupes.

6.1.1.2 Interprétation et choix des groupes pour le modèle de comportement

Le tableau 6.1 montre la répartition des individus dans les groupes formés. La première colonne indique le groupe concerné, la deuxième donne le nombre d'individus constituant chaque groupe. La troisième colonne donne la proportion d'individus représentée par un groupe. Suivent le nombre total de requêtes effectuées vers les individus de chaque groupe et la proportion de requêtes que cela représente. Pour chaque sous-ensemble est indiqué le nombre total d'individus qu'il contient, la proportion que cela constitue, ainsi que le nombre de requêtes total vers les individus de ce sous-ensemble et la proportion que cela représente par rapport au nombre total de requêtes

Groupe	Nb d'indiv.	Pourcentage	Nb de req.	Pourcentage
1.1	3 073	4,54 %	12 501	1,28 %
1.2	6 556	9,69 %	181 595	18,65 %
1.3	1 168	1,73 %	7 693	0,79 %
1.4	2 393	3,54 %	9 768	1,00 %
1.5	1 110	1,64 %	1 644	0,17 %
1.6	7 165	10,60 %	21 371	2,20 %
1.7	719	1,06 %	1 190	0,12 %
1.8	626	0,93 %	4 254	0,44 %
1.9	40	0,06 %	876	0,09 %
Total	22 850	33,79 %	240 892	24,74 %
2.1	2 593	3,83 %	14 618	1,50 %
2.2	1 745	2,58 %	22 676	2,33 %
2.3	1 369	2,02 %	9 907	1,02 %
2.4	914	1,35 %	1 115	0,11 %
2.5	7 804	11,54 %	50 959	5,23 %
2.6	3 679	5,44 %	76 588	7,87 %
2.7	222	0,33 %	2 580	0,27 %
2.8	3 487	5,16 %	294 597	30,26 %
2.9	361	0,53 %	1 955	0,20 %
Total	22 174	32,78 %	474 995	48,79 %
3.1	2 208	3,27 %	8 605	0,88 %
3.2	1 848	2,73 %	24 454	2,51 %
3.3	9 795	14,48 %	47 523	4,88 %
3.4	1 414	2,09 %	2 330	0,24 %
3.5	5 969	8,83 %	163 031	16,75 %
3.6	612	0,90 %	9 258	0,95 %
3.7	638	0,94 %	769	0,08 %
3.8	118	0,17 %	1 661	0,17 %
Total	22 602	33,42 %	257 631	26,46 %

TAB. 6.1 – Répartition des ressources issues des données du serveur web de l'IRISA

effectuées. Ce tableau nous montre que la répartition des ressources dans les groupes de chaque sous-ensemble n'est pas homogène, de même que le volume de requête associé à chaque groupe. On observe que les groupes contenant le plus d'individus ne sont pas nécessairement ceux vers lesquels il y a le plus de requêtes (groupes 1.6, 2.5 et 3.3). On observe aussi que les groupes 1.2, 2.8 et 3.5 représentent à eux trois plus de 65% des requêtes, alors qu'ils regroupent moins de 25% des individus. La concentration de requêtes vers les groupes 1.2, 2.8 et 3.5 est un indice de la présence d'un noyau de ressources constituant l'activité principale du serveur web de l'IRISA. Parmi ces groupes se trouve par exemple la page d'accueil du serveur web. Les groupes 1.2, 2.8 et 3.5 doivent représenter l'activité habituelle de ce serveur web et devraient normalement, à la suite de notre analyse, figurer parmi les groupes de ressources utilisés pour constituer le modèle de comportement du serveur web de l'IRISA.

Les détails concernant les partitionnements effectués sont présentés dans les tableaux 6.2, 6.3 et 6.4, correspondant chacun aux trois sous-ensembles que nous avons

formés au départ. Pour chaque groupe nous avons la valeur moyenne des variables associées à chaque individu du groupe, ainsi que la variance de ces variables.

	Groupe 1.1		Groupe 1.2		Groupe 1.3		Groupe 1.4	
Variables	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**
PctFérié	0,021	0,057	0,191	0,264	0,181	1,051	0,151	1,081
Req-Jour	0,37	0,019	2,518	3,233	0,599	0,257	0,371	0,025
PctReq-Jour	4,130e-06	0,021	2,866e-05	3,202	6,891e-06	0,249	4,290e-06	0,027
PctUserId	ε	0,333	ε	0,274	0	0	0	0
PctGet	0,997	0,685	0,997	0,376	1	0,009	0,995	1,068
PctPost	0	0	ε	0,016	ε	0,018	0	0
PctHead	0,002	2,034	0,002	0,650	0	0	0,002	1,438
PctVide	0	0	0	0	0	0	0	0
PctAutresMeth	0,001	0,601	0,001	0,879	0	0	0,003	3,124
PctHTTP/1.x	1	0	1	0,009	1	16,114	1	1,654
Pct200	0,001	0,001	0,859	0,105	0,934	0,273	0,014	0,020
Pct300	ε	ε	0,141	0,182	0,004	0,014	0,986	0,036
Pct400	0,999	0,001	ε	ε	0,025	0,173	ε	ε
Pct500	0	0	ε	ε	0,037	18,739	0	0
args	0	0	ε	0,002	1	0	0	0

	Groupe 1.5		Groupe 1.6		Groupe 1.7		Groupe 1.8		Groupe 1.9	
Variables	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**
PctFérié	0,962	0,13	0,003	0,004	0,902	0,446	0,18	1,122	0,047	0,348
Req-Jour	0,135	ε	0,271	0,003	0,15	0,004	0,618	1,128	1,991	0,417
PctReq-Jour	2,072e-06	ε	2,909e-06	0,003	2,249e-06	0,007	7,722e-06	1,437	2,229e-05	0,4
PctUserId	0	0	0,004	2,789	0	0	0	0	0	0
PctGet	0,998	0,436	0,997	0,494	0,999	0,103	1	0	0,01	1,189
PctPost	0	0	0	ε	0	0	0	0	0,99	2,313
PctHead	0,002	1,599	0,002	0,955	0,001	0,379	0	0	0	0
PctVide	0	0	0	0	0	0	0	0	0	0
PctAutresMeth	0	0	0,001	1,082	0	0	0	0	0	0
PctHTTP/1.x	1	0	1	0	1	0	1	0	1	0
Pct200	0,977	0,045	0,995	0,003	0	0	0,002	0,005	0,476	1,315
Pct300	0,023	0,077	0,004	0,005	0	0	0,998	0,009	0,024	0,197
Pct400	0	0	ε	0,001	1	0	0	0	0,5	1,835
Pct500	0	0	0	0	0	0	0	0	0	0
args	0	0	0	0	0,007	0,093	1	0	1	0

TAB. 6.2 – Caractéristiques des 9 groupes résultant d'un partitionnement utilisant la méthode CLARA sur le premier sous-ensemble des données provenant du serveur de l'IRISA. Les variances sont calculées sur les variables centrées et réduites.

Premier sous-ensemble des données de l'IRISA

On observe dans le tableau 6.2 que la plupart des groupes issus du premier sous-ensemble présentent un profil similaire à ceux que nous avons établi au Chapitre 4. En effet, mis à part le groupe 1.9, tous les groupes ont une valeur tranchée concernant la méthode utilisée pour accéder aux ressources et le code de retour envoyé par le serveur. Les groupes 1.2, 1.3, 1.4, 1.5, 1.6 et 1.8 sont des groupes de ressources demandées avec la méthode `GET` et servies avec succès par le serveur web, alors que les ressources des groupes 1.1 et 1.7 n'ont engendré que des erreurs. Seules les ressources du groupe 1.9 sont demandées avec la méthode `POST`.

Seuls les groupes 1.3, 1.8 et 1.9 sont constitués de ressources dynamiques, le groupe 1.9 étant constitué de ressources dynamiques presque exclusivement demandées avec la méthode `POST`.

Concernant la proportion de requêtes effectuées les jours ouvrés, on note que les groupes 1.2, 1.3, 1.4, 1.8 ont un profil équilibré. Le serveur web de l'IRISA étant un serveur académique, il n'est pas suspect d'observer des groupes de ressources presque exclusivement demandées les jours ouvrés (c-à-d. pendant les jours de travail habituel des ensignants et des étudiants), tel que pour les groupes 1.1, 1.6 et 1.9. Les groupes 1.5 et 1.7 ne concernent que des ressources très peu demandées (resp. 1,48 requêtes par ressource et 1,66 requêtes par ressources). Le faible volume de demande des ressources des groupes 1.5 et 1.7 fausse l'interprétation que l'on pourrait faire de la forte proportion de demande les jours fériés. Nous ne considérons pas *a priori* que cette dernière caractéristique constitue un élément suspect.

Parmi ces groupes, seul le groupe 1.9 ne correspond pas à un profil que nous ayons pu établir précédemment. Les requêtes vers les ressources de ce groupe sont seulement pour moitié réussies.

Nous choisissons d'intégrer les groupes 1.2, 1.3, 1.4, 1.5, 1.6, et 1.8 au modèle de comportement. Ces groupes représentent un mode de fonctionnement normal du serveur web, c'est-à-dire des ressources demandées avec succès par les clients. Les groupes 1.1 et 1.7 sont constitués de ressources générant des erreurs. Le taux de requêtes associé aux groupes 1.1 et 1.7 (respectivement 1,28% et 0,12%) n'est pas en rapport avec la proportion de ressources qu'ils représentent (respectivement 4,54% et 1,06%). Nous ne considérons pas qu'il puisse s'agir de ressources générant des erreurs récurrentes liées à des requêtes automatisées, mais plutôt des erreurs ponctuelles. En conséquence, nous choisissons de ne pas intégrer les ressources des groupes 1.1 et 1.7 à notre modèle de comportement. Le groupe 1.9 n'entre pas dans nos profils. Il ne constitue qu'un groupe de 40 individus, vers lequel ne sont faites que 876 requêtes (soit moins de 0,1 % du total des requêtes). De plus, ce groupe est constitué essentiellement de ressources dynamiques. Dans le doute et vu le faible taux de filtrage qu'apporterait l'intégration de ce groupe, nous ne l'intégrons pas au modèle de comportement.

	Groupe 2.1		Groupe 2.2		Groupe 2.3		Groupe 2.4	
Variables	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**
PctFérié	0,146	1,344	0,153	1,185	0,508	0,121	0,999	0,002
Req-Jour	0,512	0,007	1,181	0,188	0,658	0,007	0,111	ε
PctReq-Jour	5,777e-06	0,007	1,333e-05	0,188	8,634e-06	0,01	1,754e-06	ε
PctUserId	0,002	0,086	0	0	0	0	0	0
PctGet	0,993	3,26	0,995	1,534	0,992	1,545	0,999	0,659
PctPost	0	0	0	0	0	0	0	0
PctHead	0,003	2,313	0,004	2,424	0,008	2,87	0,001	1,224
PctVide	0	0	0	0	0	0	0	0
PctAutresMeth	0,004	5,648	0,001	0,624	0	0	0	0
PctHTTP/1.x	1	0	1	0	1	0	1	0
Pct200	0,002	0,003	0,053	0,070	0,946	0,066	0,993	0,022
Pct300	0,001	0,002	0,947	0,136	0,053	0,122	0,007	0,042
Pct400	0,998	0,006	ε	0,001	0,001	0,004	0	0
Pct500	0	0	0	0	0	0	0	0
args	0	0	0	0	0	0	0	0

	Groupe 2.5		Groupe 2.6		Groupe 2.7		Groupe 2.8		Groupe 2.9	
Variables	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**
PctFérié	0,006	0,006	0,207	0,08	0,012	0,147	0,133	0,252	0,189	1,219
Req-Jour	0,594	0,005	1,893	0,151	1,057	0,018	7,68	6,001	0,492	0,002
PctReq-Jour	6,423e-06	0,005	2,176e-05	0,155	1,094e-05	0,014	8,516e-05	5,996	5,775e-06	0,003
PctUserId	0	ε	0	0	0,962	1,157	0,001	0,03	0,003	0,203
PctGet	0,998	0,443	0,998	0,391	0,989	4,064	0,998	0,216	0,992	4,788
PctPost	0	ε	0	0	0,005	24,974	0	0	0,008	45,818
PctHead	0,001	0,496	0,002	0,401	0,003	1,982	0,002	0,304	0	0
PctVide	0	0	0	0	0	0	0	0	0	0
PctAutresMeth	ε	0,493	ε	0,491	0,003	1,204	ε	0,138	0	0
PctHTTP/1.x	1	ε	1	0	1	0	1	0	0,995	61,256
Pct200	0,991	0,006	0,967	0,019	0,839	0,624	0,677	0,118	0,614	1,596
Pct300	0,01	0,011	0,033	0,036	0,063	0,413	0,322	0,227	0,321	2,829
Pct400	ε	ε	ε	ε	0,099	0,617	0,001	0,005	0,059	0,514
Pct500	0	0	0	0	0	0	0	0	0	0
args	0	0	0	0	0,014	0,829	0	0	1	0

TAB. 6.3 – Caractéristiques des 9 groupes résultant d'un partitionnement utilisant la méthode CLARA sur le deuxième sous-ensemble des données provenant du serveur de l'IRISA. Les variances sont calculées sur les variables centrées et réduites.

Deuxième sous-ensemble des données de l'IRISA

On observe dans le tableau 6.3 que les groupes formés à partir du deuxième sous-ensemble présentent aux aussi des profils identifiables. En effet, tous les groupes présentent des valeurs tranchées suivant l'utilisation de la méthode et le succès des requêtes effectuées. Toutes les ressources composant les groupes sont demandées avec la méthode `GET`. Les groupes 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8 et 2.9 contiennent les ressources demandées avec succès, contrairement au groupe 2.1. Le groupe 2.9 regroupe les ressources dynamiques demandées avec succès. Le groupe 2.7 contient les ressources dont l'accès requiert une authentification. Cette information peut expliquer le taux d'erreur (`Pct400`) d'environ 10 %.

Les groupes 2.1, 2.2, 2.6, 2.8 et 2.9 ont un profil équilibré concernant la proportion de requête effectuées les jours fériés et les jours ouvrés. Les groupes 2.5 et 2.7 contiennent les ressources demandées presque exclusivement les jours ouvrés. De même que pour les groupes 1.5 et 1.7, le groupe 2.4 contient des ressources très peu demandées (1,22 requêtes par ressources en moyenne). Seul le groupe 2.3 montre un déséquilibre que nous ne savons pas expliquer (parité entre les requêtes effectuées les jours ouvrés et les jours fériés). Cependant, le faible nombre de requêtes par ressource (environ 7,24 requêtes par ressources) ne nous permet pas d'interpréter ce déséquilibre. Nous ne tenons donc pas compte de cette caractéristique pour la sélection du groupe 2.3.

Les groupes formés à partir du deuxième sous-ensemble des données du serveur web de l'IRISA correspondent tous à des profils que nous avons établis précédemment et le choix de leur intégration au modèle de comportement ne pose pas de problème. Nous intégrons tous les groupes formés, sauf le groupe 2.1.

	Groupe 3.1		Groupe 3.2		Groupe 3.3		Groupe 3.4	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,026	0,067	0,177	1,15	0,007	0,007	0,946	0,156
Req-Jour	0,354	0,008	1,203	0,254	0,441	0,006	0,15	ε
PctReq-Jour	3,991e-06	0,007	1,375e-05	0,225	4,747e-06	0,005	2,272e-06	ε
PctUserId	ε	0,452	0	0	ε	0,079	0	0
PctGet	0,993	0,678	0,997	0,296	0,996	0,281	0,968	3,277
PctPost	0	0	0	0	0	ε	0	0
PctHead	0,003	0,793	0,001	0,2	0,003	0,457	0,031	10,174
PctVide	0	0	0	0	0,001	1,774	0,001	2,459
PctAutresMeth	0,004	4,971	0,002	2,671	ε	0,361	0	0
PctHTTP/1.x	0,999	1,919	1	0	0,999	1,442	0,999	1,998
Pct200	0,003	0,006	0,055	0,095	0,979	0,032	0,986	0,042
Pct300	ε	0,001	0,944	0,19	0,019	0,05	0,014	0,076
Pct400	0,996	0,012	0,001	0,001	0,001	0,004	0	0
Pct500	ε	0,31	0	0,001	0,001	1,869	0,001	1,934
args	0	0	0	0	0	0	0	0

	Groupe 3.5		Groupe 3.6		Groupe 3.7		Groupe 3.8	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,276	0,272	0,152	0,853	0,956	0,212	0,008	0,027
Req-Jour	2,483	3,595	1,375	0,768	0,11	ε	1,28	0,04
PctReq-Jour	2,866e-05	3,608	1,581e-05	0,748	1,680e-06	ε	1,359e-05	0,032
PctUserId	0,002	3,487	0	0	0	0	0	0
PctGet	0,999	0,038	0,995	0,332	0,994	0,671	0,014	0,434
PctPost	0	0	0,003	0,274	0	0	0,986	0,79
PctHead	0,001	0,092	0,002	0,594	0,002	0,548	0	0
PctVide	0	ε	0	0	0,002	2,727	0	0
PctAutresMeth	ε	0,113	0	0	0,003	3,831	0	0
PctHTTP/1.x	1	ε	1	0	0,998	2,216	1	0
Pct200	0,876	0,171	0,662	1,411	0	0	0,872	0,625
Pct300	0,123	0,338	0,296	2,619	0	0	0,064	0,734
Pct400	ε	0,001	0,041	0,355	1	0	0,051	0,392
Pct500	ε	ε	0,001	0,205	0	0	0,012	6,034
args	0	0	1	0	0	0	1	0

TAB. 6.4 – Caractéristiques des 8 groupes résultant d'un partitionnement utilisant la méthode CLARA sur le troisième sous-ensemble des données provenant du serveur de l'IRISA. Les variances sont calculées sur les variables centrées et réduites.

Troisième sous-ensemble des données de l'IRISA

Les groupes formés à partir des données du troisième sous-ensemble de données du serveur web de l'IRISA, présentés dans le tableau 6.4, présentent eux aussi des profils identifiables. Tous les groupes présentés dans le tableau 6.4 contiennent des ressources demandées avec la méthode `GET`, sauf le groupe 3.8 dont les ressources sont demandées avec la méthode `POST`. Les groupes 3.2, 3.3, 3.4, 3.5, 3.6 et 3.8 contiennent les ressources demandées avec succès, contrairement aux groupes 3.1 et 3.7. Les groupes 3.6 et 3.8 sont constitués de ressources dynamiques.

Seuls les groupes 3.2, 3.5 et 3.6 ont un profil équilibré concernant la répartition des requêtes vers les ressources en fonction des jours ouvrés et jours fériés. Les groupes 3.4 et 3.7 qui peuvent paraître problématiques sont composés de ressources là aussi très peu demandées (respectivement 1,58 et 1,26 requêtes par ressources en moyenne).

Les groupes formés à partir du troisième sous-ensemble des données de l'IRISA sont facilement interprétables à partir des profils déjà établis. Nous choisissons d'intégrer les groupes 3.2, 3.3, 3.4, 3.5, 3.6 et 3.8 et de ne pas sélectionner les groupes 3.1 et 3.7.

Le tableau 6.5 récapitule les profils et caractéristiques des groupes formés à partir des données du serveur web de l'IRISA. Les groupes que nous avons choisi d'intégrer au modèle de comportement sont en gras. On remarque que le profil « `GET` + succès » est prépondérant dans le choix de l'intégration des groupes.

Les vingt groupes que nous intégrons au modèle de comportement contiennent 58 355 ressources, soit 86,29 % des ressources présentes dans notre corpus d'apprentissage. Les ressources intégrées au modèle de comportement ont généré 934 559 requêtes soit 96,04 % du total des requêtes émises vers le serveur web durant la période d'apprentissage.

6.1.1.3 Analyse de la qualité du modèle de comportement

Complétude

Si les ressources présentes sur le serveur web sont effectivement pérennes, nous pouvons estimer le taux de filtrage du module de détection d'intrusions comportementale à plus de 96 %. Cependant, notre période d'apprentissage n'est constituée que de 10 jours de trafic vers le serveur web de l'IRISA. Même si le volume de trafic nous a semblé assez important sur cette période pour effectuer un partitionnement des ressources demandées sur cette période, cette courte durée peut impliquer que de nombreuses ressources n'aient pas été demandées pendant la période d'observation que nous avons choisie. De plus, il nous faut prendre en compte l'évolution du contenu du serveur web de l'IRISA. On sait que la publication de nouvelles ressources n'est pas centralisée, mais que de nombreux utilisateurs (étudiants, professeurs, ...) peuvent eux-mêmes publier de nouvelles ressources. Pour ces deux raisons, nous pouvons difficilement considérer que le taux de requêtes associé aux ressources composant le modèle de comportement puisse refléter le taux de filtrage que nous obtiendrons en pratique. Dans ce contexte, nous ne soncidérons pas que la complétude du modèle de comportement soit bonne. Le taux de filtrage de *WebFilter* risque d'être effectivement assez éloigné de nos attentes, contrairement aux résultats obtenus pour le serveur de Supélec. Les résultats expérimentaux

Profils	**Nom**	**Groupes**
Méthode + Réponse	`GET` + succès	**1.2**, **1.3**, **1.4**, **1.5**, **1.6**, **1.8** **2.2**, **2.3**, **2.4**, **2.5**, **2.6**, **2.7**, **2.8** **3.2**, **3.3**, **3.4**, **3.5**, **3.6**
	`GET` + échec	1.7, 2.1, 3.1, 3.7
	« autre »	1.1 (`POST`), 1.9, **2.9**, **3.8**
Répartition	équilibrée	**1.2**, **1.3**, **1.4**, **1.5**, **1.8** 2.1, **2.2**, **2.6**, **2.7**, **2.8**, **2.9** **3.2**, **3.5**, **3.6**
	tranchée	1.1, **1.6**, 1.7, 1.9 **2.4**, **2.5** 3.1, **3.3**, **3.4**, 3.7, **3.8**
	« autre »	**2.3**
Volumétrie	forte	**2.7**
	moyenne	**1.2**, 1.9 **2.2**, **2.6** **3.2**, **3.5**, **3.6**, **3.8**
	faible	1.1, **1.3**, **1.4**, **1.5**, **1.6**, 1.7, **1.8** 2.1, **2.3**, **2.4**, **2.5**, **2.8**, **2.9** 3.1, **3.3**, **3.4**, 3.7

TAB. 6.5 – Profils des groupes issus de la classification

présentés à la Section 6.3.2 illustrent les carences du modèle de comportement que nous avons établi.

Correction

Nous analysons manuellement le contenu de chaque groupe formé par l'analyse statistique afin d'évaluer la pertinence de notre sélection en fonction des ressources cibles d'attaques. Le tableau 6.6 décompte, pour chaque groupe, le nombre de ressources contenues dans le groupe et le nombre de ressources cibles d'attaques.

Parmi les groupes sélectionnés pour faire partie du modèle de comportement, les groupes 1.3, 3.3, 3.4 et 3.8 contiennent des ressources susceptibles d'être des cibles d'attaques. Le groupe 1.3 contient des traces de *scans* vers le script `FormMail` et vers une extension FrontPage, ainsi qu'une tentative d'accès échouée vers un script utilisé comme compteur. Le groupe 3.3 contient 12 ressources suspectes. Parmi celles-ci, quatre semblent être des tentatives échouées de détournement de ressources liées à l'envoi de *spam*. Cinq autres ressources sont simplement constituées de quelques caractères de contrôle et ne semblent être que des erreurs de manipulation de la part du client. Les trois dernières ressources ne sont pas des attaques à proprement parler, mais elles dénotent, pour le serveur web, un mode de fonctionnement suspect. En effet, il s'agit d'URL complètes (par exemple `http://www.monsite.com/`), ayant été traitées avec succès par le serveur web de l'IRISA, ce qui montre que le serveur peut être utilisé

Groupe	Nb de res.	Nb de res. cibles d'attaques	Groupe	Nb de ress.	Nb de res. cibles d'attaques
1.1	3 073	0	2.1	2 593	0
1.2	**6 556**	**0**	**2.2**	**1 745**	0
1.3	**1 168**	6	**2.3**	**1 369**	0
1.4	**2 393**	0	**2.4**	**914**	0
1.5	**1 110**	0	**2.5**	**7 804**	0
1.6	**7 165**	0	**2.6**	**3 679**	0
1.7	719	0	**2.7**	**222**	0
1.8	**626**	0	**2.8**	**3 487**	0
1.9	40	17	**2.9**	**361**	0
3.1	2 208	7			
3.2	**1 848**	0			
3.3	**9 795**	12			
3.4	**1 414**	2			
3.5	**5 969**	0			
3.6	**612**	0			
3.7	638	3			
3.8	**118**	1			

TAB. 6.6 – Analyse des ressources composant les groupes

comme proxy par des clients. Les deux ressources suspectes contenues dans le groupe 3.4 concernent là aussi une tentative d'envoi de *spam* et une ressource dont le nom est composé de quelques caractères de contrôle. Les requêtes liées à ces ressources ont toutes échouées. Le groupe 3.8 contient, comme le groupe 1.3, une tentative d'accès vers un script `FormMail` qui a échouée.

Aucune des tentatives d'accès vers des cibles d'attaque n'a eu de succès. Cela signifie que soit les ressources cibles d'attaques ne sont pas présentes sur le serveur, soit elles ont été mises à jour et ne constituent plus un danger. L'intégration de ces ressources au modèle de comportement aura pour seule conséquence de masquer des tentatives d'accès infructueuses, à moins que les dites ressources vulnérables ne soient effectivement hébergées sur le serveur ultérieurement.

Parmi les groupes que nous avons exclus du modèle de comportement, seuls les groupes 1.9, 3.1 et 3.7 contiennent des ressources susceptibles d'être des cibles d'attaque. Le groupe 1.9 contient 17 ressources cibles d'attaques potentielles sur les 40 ressources que compte le groupe. On dénombre quatre ressources liées à des extensions FrontPage, six ressources liées à des tentatives d'accès vers un script `FormMail`, les sept autres ressources étant le résultat de *scan* vers des scripts n'existant pas sur le serveur. Le groupe 3.1 compte 7 ressources suspectes. Une de ces ressources est une tentative de *buffer overflow* contre le serveur web. Les autres ressources sont liées à une utilisation du serveur web comme proxy. Concernant ces requêtes, le serveur web est utilisé de façon frauduleuse pour transmettre des requêtes HTTP, mais aussi des

requêtes SMTP. Le protocole SMTP sert à l'envoi de mail, et l'utilisation du serveur web comme proxy permet par exemple à un émetteur de *spam* de se « cacher » derrière le serveur web de l'IRISA. Le groupe 3.7 contient 3 ressources suspectes. Une de ces ressources est composée de quelques caractères de contrôle et semble être une erreur de manipulation du client. Les deux autres ressources concernent là encore l'utilisation du serveur web de l'IRISA comme proxy vers un serveur SMTP.

Les groupes 1.1, 1.7 et 2.1 ne contiennent aucune attaque mais contiennent des ressources générant des erreurs, ce qui indique que ces ressources ne sont pas sur le serveur. Le faible volume de requêtes vers ces ressources nous laisse à penser qu'il s'agit soit d'erreurs lors de la saisie de l'URL par le client, soit de ressources supprimées du serveur vers lesquelles pointent encore quelques liens.

Conclusion

Dans cette partie, nous avons appliqué la méthodologie que nous avons présentée au Chapitre 4 sur un autre serveur web académique, le serveur web de l'IRISA. Pour appréhender le trop grand volume de ressources, nous avons scindé le corpus d'apprentissage de l'IRISA en trois sous-ensembles. Cette scission nous a permis de les considérer comme trois sites distincts et d'appliquer notre analyse statistique sur chacun des trois sous-ensemble. Finalement nous avons fusionné les résultats obtenus afin de constituer le modèle de comportement.

Les groupes que nous avons obtenus montrent que les premiers profils que nous avons établis à partir des données du serveur web de Supélec s'appliquent aussi en partie aux résultats obtenus avec les données du serveur web de l'IRISA. L'interprétation des groupes formés se fait assez simplement, suivant le mode opératoire que nous avions adopté avec les groupes issus des données du serveur web de Supélec.

Les résultats obtenus nous permettent de sélectionner 86,29 % des ressources du corpus d'apprentissage, soit 58 355 ressources. Cependant, la courte durée d'apprentissage ne nous permet pas d'augurer le taux de filtrage apporté par les ressources sélectionnées.

L'analyse manuelle des ressources intégrées au modèle de comportement montre que celui-ci ne contient pas d'attaques réussies, mais des tentatives d'accès issues de *scans*. Le modèle que nous avons formé n'engendrera *a priori* pas de faux négatifs.

6.1.2 Un serveur web de France Télécom

Le dernier serveur web que nous utilisons en test est un serveur web interne à France Télécom. Nous utilisons 7 jours de log, du 16 au 22 octobre 2004 inclus. Les données que nous analysons sont constituées de 2 722 457 requêtes vers 925 ressources, soit près de 3 000 requêtes par ressources et environ 389 000 requêtes par jour.

On observe que contrairement aux serveurs web de Supélec et de l'IRISA, le serveur web de France Télécom n'a servi que peu de ressources différentes à ses clients durant la période considérée (925 ressources) alors que le nombre de requêtes par ressource est sans commune mesure. Cela vient du fait que ce serveur est un serveur destiné aux personnels de France Télécom et dont la publication de ressources est gérée par une

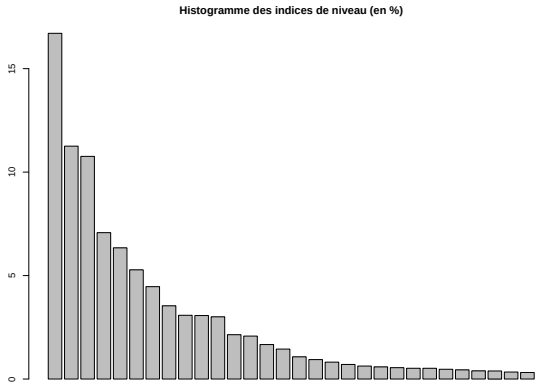

FIG. 6.4 – Histogramme des indices de niveaux pour la CAH réalisées sur les données du serveur web de France Télécom

équipe dédiée.

Ici, c'est le nombre de requêtes qui limite notre durée d'observation. En effet nous atteignons les limites de traitements de nos machines et nous restreignons à une période d'observation de 7 jours. Cette courte période nous permet malgré tout d'observer le comportement des clients sur les jours ouvrés et fériés. Cependant, nous devons tenir compte dans notre analyse de cette courte durée d'observation, plus particulièrement sur l'efficacité et la pérennité du modèle de comportement que nous créons.

6.1.2.1 Choix du nombre de groupes à former

Le faible nombre de ressources à traiter nous permet de nous affranchir de l'échantillonnage utilisé sur les ressources de Supélec et de l'IRISA. Nous pouvons effectuer une CAH en utilisant l'ensemble des ressources du serveur web de France Télécom. La Figure 6.4 nous montre la courbe d'indice de niveau résultant de la CAH. Nous observons une stabilisation nette entre le partitionnement en onze groupes et le partitionnement en neuf groupes et une forte dissimilarité pour le passage à un partitionnement en dix groupes. Nous choisissons de partitionner les ressources du serveur web de France Télécom en 9 groupes.

6.1.2.2 Interprétation et choix des groupes pour le modèle de comportement

Le grand nombre de ressources issues des serveurs web de Supélec et de l'IRISA nous a poussé à utiliser la méthode CLARA pour effectuer le partitionnement de ces ressources. Dans le cas du serveur web de France Télécom, le nombre de ressources à partitionner nous autorise à utiliser directement la méthode PAM (cf. Chapitre 3) en lieu et place de la méthode CLARA.

Groupe	Nb ressources	Pourcentage	Nb de requêtes	Pourcentage
1	103	11,14 %	4 845	0,18 %
2	23	2,49 %	2 294 250	84,27 %
3	38	4,10 %	47 243	1,74 %
4	117	12,65 %	1 308	0,05 %
5	131	14,16 %	6 258	0,23 %
6	123	13,30 %	2 817	0,10 %
7	23	2,49 %	2 167	0,08 %
8	247	26,70 %	353 806	13,00 %
9	120	12,97 %	9 763	0,36 %
Total	925	100,00 %	2 722 457	100,00 %

TAB. 6.7 – Répartition des ressources issues du serveur web de France Télécom

Le tableau 6.7 montre la répartition des ressources dans les 9 groupes formés. Pour chaque groupe est indiqué le nombre de ressources qu'il contient ainsi que la proportion de ressources que cela représente par rapport à l'ensemble, le nombre de requêtes émises vers les ressources d'un groupe et la proportion de requêtes par rapport au nombre total de requêtes. Le tableau 6.7 nous montre que les ressources ne sont pas réparties de façon homogène. De même, le volume de requêtes associé à chaque groupe est lui-même très hétérogène. En effet, plus de 84 % des requêtes sont effectuées vers les ressources du groupe 2, alors que ce groupe ne contient que 2,49 % des ressources. Comme pour le serveur de l'IRISA, cela nous montre que le groupe 2 contient des ressources reflétant l'activité habituelle du serveur web de France Télécom. Là encore, ce groupe devrait finalement figurer parmi les groupes de ressources sélectionnés pour intégrer le modèle de comportement.

Le partitionnement des ressources issues du serveur web de France Télécom est présenté plus en détail dans le tableau 6.8.

	Groupe 1		Groupe 2		Groupe 3		Groupe 4		Groupe 5	
Variables	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**
PctFérié	0,93	0,163	0,417	0,003	0,639	1,073	0,04773	0,126	0,919	0,091
Req-Jour	6,72	ε	14250	0,486	177,605	0,041	1,5971	7,711e-06	6,824	ε
PctReq-Jour	1,396e-05	5,072e-05	0,037	16,318	4,658e-04	0,02	4,667e-06	6,358e-06	1,592e-05	6,646e-05
PctUserId	0,025	1,355	0	0	0,183	9,355	0	0	0,004	0,117
PctGet	0,998	0,01	0,998	0,001	0,787	5,084	0,9982	0,015	0,984	0,387
PctPost	0	0	0	0	0,200	19,285	0	0	0	0
PctHead	0,002	0,016	0,002	0,002	0,013	0,155	0,001821	0,023	0,004	0,056
PctVide	0	0	0	0	0	0	0	0	0	0
PctAutresMeth	0	0	ε	ε	0	0	0	0	0,011	3,427
PctHTTP/1.x	1	0	1	0	1	0	1	0	1	0
Pct200	0,994	0,01	0,161	0,319	0,676	1,13	0	0	0,008	0,015
Pct300	0,003	0,003	0,838	0,44	0,054	0,122	0	0	0,008	0,023
Pct400	0,004	ε	0,001	ε	0,264	0,972	1	0	0,984	0,034
Pct500	ε	ε	0	0	0,005	12,569	0	0	0	0
args	0	0	0	0	1	0	0	0	0	0

	Groupe 6		Groupe 7		Groupe 8		Groupe 9	
Variables	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**
PctFérié	0,182	0,377	0,423	0,186	0,444	0,522	0,748	0,299
Req-Jour	3,272	2,563e-05	13,46	9,641e-05	204,63	0,028	11,623	9,724e-05
PctReq-Jour	1,494e-05	6,807e-05	2,491e-04	3,974e-03	4,255e-04	0,01	2,601e-05	4,319e-05
PctUserId	0	0	0	0	0,016	0,946	0,017	1,012
PctGet	0,982	0,229	0,185	1,473	0,998	0,044	1	0
PctPost	0	0	0	0	0	0	0	0
PctHead	0,018	0,353	0,771	3,833	0	0	0	0
PctVide	0	0	0	0	0	0	0	0
PctAutresMeth	0	0	0,043	16,091	0,002	0,429	0	0
PctHTTP/1.x	1	0	1	0	1	0	1	0
Pct200	0,991	0,014	0,277	0,851	0,254	0,097	0,598	0,063
Pct300	0,005	0,009	0,016	0,017	0,745	0,13	0,391	0,09
Pct400	0,003	0,004	0,707	0,722	0,001	0,001	0,010	0,023
Pct500	0,002	3,639	0	0	0	0	0	0
args	0	0	0	0	0	0	0	0

Tab. 6.8 – Caractéristiques des 9 groupes résultant d'un partitionnement utilisant la méthode CLARA (source NETRD). Les variances sont calculées sur les variables centrées et réduites

Concernant le partitionnement des ressources du serveur web de France Télécom, nous observons des profils de groupes tels que nous en avons établis au Chapitre 4. On observe que les groupes 1, 2, 4, 5, 6, 8 et 9 ont une valeur tranchée pour la méthode utilisée et la réponse renvoyée par le serveur web. Les groupes 1, 2, 6, 8 et 9 contiennent des ressources demandées avec la méthode `GET` et servies avec succès par le serveur web. Les ressources des groupes 4 et 5 sont demandées avec la méthode `GET` mais ont engendré des erreurs.

Les groupes 3 et 7 ne correspondent à aucun profil préalablement établi. Les ressources du groupe 3 sont demandées avec les méthodes `GET` (79 %) et `POST` (20 %). Elles occasionnent des réponses positives dans la majorité des cas (73 %), mais aussi un grand nombre d'erreurs (27 %). On remarque que la proportion de requêtes effectuées avec la méthode `GET` (resp. `POST`) est équivalente à la proportion de requêtes servies avec succès (resp. ayant engendré une erreur) par le serveur web. Le groupe 7 est demandé avec les méthodes `GET` (18 %) et `HEAD` (77 %) et engendre un majorité de réponses négatives (70 %), mais aussi des réponses positives (29 %). Comme pour le groupe 3, on remarque que la proportion de requêtes effectuées avec la méthode `GET` (resp. `HEAD`) est du même ordre que la proportion de requêtes servies avec succès (resp. ayant engendré une erreur) par le serveur web.

Les ressources dynamiques sont exclusivement regroupées dans le groupe 3. De plus, ce groupe ne contient que des ressources dynamiques, demandées avec les méthodes `GET` et `POST`.

Nous ne pouvons pas interpréter la répartition des requêtes entre les jours fériés et ouvrés vers les ressources du serveur web de France Télécom comme nous l'avons fait pour les serveurs de Supélec et de l'IRISA. En effet, le serveur web de France Télécom que nous analysons est un serveur interne dont l'usage est destiné aux seuls salariés de France Télécom. La consultation des ressources devrait donc ce faire *a priori* en très grande majorité les jours ouvrés. Cependant, de nombreux utilisateurs peuvent accéder à ces ressources depuis l'extérieur, donc en dehors des horaires habituels de travail, grâce à un accès VPN. Nous devons donc rester prudents quant aux conclusions que nous pourrions tirer de la répartition temporelle des requêtes vers les ressources de chaque groupe. On observe d'ailleurs un profil équilibré tel que nous l'avons établi avec les données des serveur de Supélec et de l'IRISA seulement pour le groupe 6. Les groupes 1, 4 et 5 ont un profil tranché, les ressources des groupes 1 et 5 (resp. 4) étant quasiment exclusivement demandées les jours fériés (resp. ouvrés). De façon moins tranchés, les ressources du groupe 9 sont majoritairement demandées les jours fériés. Pour les ressources des groupes 2, 3, 7 et 8, les demandes sont quasiment faites de façon équivalente entre les jours fériés et les jours ouvrés. Le profil temporel étant difficilement interprétable, nous ne l'utilisons pas dans la sélection des groupes à intégrer au modèle de comportement.

Nous choisissons d'intégrer directement les groupes 1, 2, 6, 8 et 9 au modèle de comportement. En effet, ces groupes représentent le mode de fonctionnement normal du serveur web, c'est-à-dire qu'ils contiennent des ressources demandées avec succès par les clients. *A contrario*, nous excluons les groupes 4 et 5 qui ne représentent que des requêtes vers des ressources engendrant des erreurs. Le faible nombre de requêtes que représente

Profils	**Nom**	**Groupes**
Méthode + Réponse	`GET` + succès	1, 2, 6, 8, 9
	`GET` + échec	4, 5
	« autre »	3, 7
Volumétrie	forte	2, 8
	moyenne	3
	faible	1, 4, 5, 6, 7, 9

TAB. 6.9 – Profils des groupes issues de la classification

ces groupes nous permet de considérer qu'il ne s'agit pas de ressources générant des erreurs récurrentes, mais plutôt des erreurs ponctuelles. Seuls les groupes 3 et 7 ont des profils que nous ne pouvons pas interpréter directement. Même si le succès des requêtes émises vers les ressources du groupe 3 est moins tranché que pour les groupes 1, 2, 6, 8 et 9, on observe que les deux tiers des requêtes sont servies avec succès par le serveur. De plus, le groupe 3 contient toutes les ressources dynamiques demandées et on sait que le serveur web contient des ressources dynamiques. Finalement, le serveur web de France Télécom que nous étudions est un serveur interne, donc normalement moins sujet aux attaques que les serveurs web accessibles depuis internet. Nous choisissons donc d'intégrer les ressources du groupe 3 au modèle de comportement. Les ressources du groupe 7 sont demandées en majorité avec la méthode `HEAD`. Cette méthode peut être utilisée dans des *scans* visant à établir la présence d'une ressource vulnérable sur le serveur web. Cependant, l'utilisation de la méthode `HEAD` correspond plus certainement à des requêtes relatives à la mise à jour de cache de proxys par exemple. La proportion de requêtes réussies vers les ressources du groupe 7, et l'absence de ressources dynamiques nous décide à intégrer ce groupe de ressources au modèle de comportement.

Le tableau 6.9 récapitule les profils et caractéristiques des groupes formés à partir des données du serveur web de France Télécom. Nous ne représentons pas les profils liés à la répartition des demandes des ressources entre les jours ouvrés et les jours fériés, ne sachant pas comment interpréter les valeurs fournies dans le contexte d'un serveur web interne. Le tableau 6.10 récapitule les choix d'intégration que nous avons fait ainsi que le nombre de ressources dans chaque groupe, et le nombre requêtes faites vers les ressources de chaque groupe durant la période analysée. Nous intégrons sept groupes de ressources sur les neuf groupes formés.

Les sept groupes que nous intégrons au modèle de comportement représentent 677 ressources, soit 73,19 % du total des ressources du corpus d'apprentissage. Les ressources intégrées au modèle de comportement ont générées 2 714 891 requêtes, soit 99,73 % du total des requêtes émises vers le serveur web durant la période d'apprentissage.

6.1.2.3 Analyse de la qualité du modèle de comportement

Complétude

Concernant la complétude du modèle de comportement que nous venons de consti-

Groupe	Intégré	Nb res.	Nb req.
1	oui	103	4 845
2	oui	23	2 294 250
3	oui	38	47 143
4	non	117	1 308
5	non	131	6 258
6	oui	123	2 817
7	oui	23	2 167
8	oui	247	353 806
9	oui	120	9 763

TAB. 6.10 – Récapitulatif des groupes intégrés au modèle de comportement

tuer, nous faisons les mêmes réserves que celles émises pour le modèle de comportement du serveur web de l'IRISA. En effet, le corpus d'apprentissage n'est constitué que de sept jours d'activités et cette courte période d'apprentissage ne nous permet pas d'augurer de la pérennité des ressources constituant le modèle de comportement. Nous considérons donc qu'*a priori*, le taux de requêtes associé aux ressources composant le modèle de comportement du serveur web de France Télécom (99,73 %) peut ne pas refléter le taux de filtrage que nous obtiendrons. En pratique, les expérimentations présentés à la Section 6.3.3 montrent que le filtrage effectué par *WebFilter* est très efficace.

Correction

Le modèle de comportement formé à partir des données du serveur de France Télécom ne contient aucune ressource cible d'attaques. Plus globalement, l'analyse complète du corpus provenant du serveur web de France Télécom ne contenait pas de ressources cibles d'attaques. Cette caractéristique s'explique par le fait que ce serveur est un serveur web interne, il n'est donc pas soumis aux différents scans et à l'activité virale qui constitue le bruit de fond de l'activité intrusive sur Internet. De plus, on peut considérer qu'*a priori* les machines appartenant au réseau de France Télécom sont saines puisque gérées par une équipe dédiée. Cependant, ce contexte n'affranchit pas le serveur web de France Télécom d'éventuelles attaques menées de l'intérieur.

Conclusion

Dans cette section, nous avons construit le modèle de comportement d'un serveur web interne à France Télécom. Contrairement aux serveurs web de Supélec et de l'IRISA, nous avons utilisé l'algorithme PAM afin de créer les groupes de ressources, au lieu d'utiliser l'algorithme CLARA.

Pour la formation des groupes, nous avons seulement utilisé les profils relatifs au couple « Méthode + Réponse ». En effet, le déséquilibre observé dans la volumétrie ne nous donne pas d'indications, sauf pour le groupe 2 dont les ressources génèrent près de

85 % des requêtes. De même, la répartition des requêtes entre les jours fériés et ouvrés n'est pas utilisée à cause des conditions d'accès particulières au serveur web de France Télécom.

Nous incluons 677 ressources dans le modèle de comportement, soit 73,19 % du total des ressources du corpus d'apprentissage. Comme pour la création du modèle de comportement du serveur web de l'IRISA, la courte période d'apprentissage (7 jours) ne nous autorise pas à tirer des perspectives concernant les capacités de filtrage du modèle de comportement ainsi créé.

L'analyse manuelle des ressources composant le modèle de comportement montre qu'il n'y a aucune ressource cible d'attaques connues, réussies ou échouées.

6.2 Résultats d'analyse de *WebAnalyzer*

Nous présentons dans cette section les résultats obtenus avec *WebAnalyzer* seul. *WebAnalyzer* est utilisé avec les mêmes signatures sur les trois serveurs web présentés précédemment (Supélec, IRISA et France Télécom).

Pour chacun des serveurs web, nous présentons les résultats obtenus avec *WebAnalyzer* sous la forme de tableaux. La première colonne indique les différents niveaux de sévérité. La deuxième colonne indique le nombre de requêtes. La troisième colonne nous montre le pourcentage de requêtes pour chaque niveau de sévérité. La quatrième colonne indique le nombre de ressources différentes vers lesquelles sont effectuées les requêtes. La dernière colonne indique le pourcentage de ressources.

6.2.1 Le serveur web de Supélec

Nous analysons avec *WebAnalyzer* 30 jours consécutifs de log du serveur web de Supélec (avril 2003). Le corpus analysé est composé de 794 647 requêtes vers 26 170 ressources. Les résultats obtenus sont présentés dans le tableau 6.11.

Parmi les requêtes de sévérité nulle, on retrouve en très grande majorité des requêtes vers des ressources statiques. Seules deux ressources sont des ressources dynamiques. On ne dénombre aucune attaque parmi les requêtes ayant une sévérité nulle.

L'ensemble des requêtes ayant sévérité 1 à 4 sont majoritairement des faux positifs. La plupart des faux positifs sont dus à la présence d'encodage non ASCII dans la requête. On dénombre cependant des tentatives d'accès échouées vers des scripts CGI relatifs à l'envoi de *spam*. On observe aussi des tentatives de connexions vers des extensions propres au serveur IIS. Ces tentatives ne sont pas nécessairement des attaques puisqu'elles peuvent résulter de l'action de *plugins* du navigateur Internet Explorer de Microsoft. Le serveur web de Supélec étant un serveur Apache, ces tentatives de connexions ont toutes échouées. On observe aussi parmi les requêtes, des tentatives (légitimes ou pas) de listage des répertoires. Ces requêtes visent à révéler l'ensemble des ressources (fichiers ou répertoires) présentes dans un répertoire donné. Dans les requêtes de sévérité 3 et 4, on observe quelques requêtes relatives à l'activité des vers Nimda et CodeRedII. Toutes les requêtes suspectes ont échouées. Pour les niveaux de sévérité 1 à 4, on dénombre 104 ressources cibles d'attaques.

Sévérité	Nb requêtes	Pourcentage	Nb ressources	Pourcentage
0	627 123	78,9184%	18 855	72,048 %
1	96 443	12,1366%	5 102	19,496 %
2	59 142	7,4425%	1 421	5,430 %
3	6 425	0,8085%	463	1,769 %
4	5 045	0,6349%	223	0,852 %
5	187	0,0235%	85	0,324 %
6	120	0,0151%	13	0,050 %
7	4	0,0005%	4	0,015 %
8	0	0,0000%	0	0,000 %
9	0	0,0000%	0	0,000 %
> 9	158	0,0199%	4	0,015 %
Total	794 647	100,0000%	26 170	100,000 %

TAB. 6.11 – Analyse des données de Supélec avec *WebAnalyzer* seul pour la période du mois d'avril 2003 (30 jours)

Concernant les requêtes marquées d'une sévérité entre 5 et 9, nous n'observons que quelques faux positifs. Ils sont dus à l'utilisation d'encodage non ASCII dans la requête ou à l'utilisation de la méthode HTTP `CONNECT` qui est suspecte (Bugtraq ID 4131). Toutes les autres requêtes résultent de l'activité du vers Nimda. Pour les niveaux de sévérité 5 à 9, on dénombre 12 ressources cibles d'attaques.

Parmi les ressources engendrant des alertes de niveau supérieur ou égal à 10, trois sont des tentatives de *buffer overflow*, résultant probablement de l'activité du vers CodeRedII. La quatrième ressource est une ressource dynamique légitime du serveur utilisant des caractères non ASCII dans la valeur de ses paramètres ainsi que des mots-clés considérés comme suspects par *WebAnalyzer*.

6.2.2 Le serveur web de l'IRISA

Dans cette section, nous analysons 28 jours consécutifs du serveur web de l'IRISA (octobre 2004). Le corpus analysé est composé de 2 981 061 requêtes vers 128 910 ressources. Le tableau 6.12 présente les résultats obtenus.

Comme pour le serveur de Supélec, les requêtes de sévérité nulle sont quasiment toutes des requêtes vers des ressources statiques. On dénombre seulement quelques requêtes vers des ressources dynamiques. Après analyse manuelle, il s'avère que ce sont des requêtes mal formées, certainement dues à des erreurs lors de la publication des ressources. Ces requêtes ne contiennent aucun élément suspect. En revanche, on dénombre des requêtes visant à utiliser le serveur web de l'IRISA comme proxy (requêtes du type `GET http ://site.web.cible.com/ HTTP/1.1`). En effet, *WebAnalyzer* ne dispose pas de signature permettant de révéler ce genre d'activité. Une signature détectant la présence de la chaîne `http ://` en tête de l'URL permettrait de détecter cette activité. Ces requêtes ont toutes échouées et ne représentent pas un risque pour le serveur

Sévérité	Nb requêtes	Pourcentage	Nb ressources	Pourcentage
0	2 310 108	77,4928 %	80 319	62,306%
1	573 164	19,2268 %	31 841	24,700%
2	69 635	2,3359 %	11 972	9,287%
3	19 328	0,6484 %	3 164	2,454%
4	8 533	0,2862 %	1 526	1,184%
5	109	0,0037 %	26	0,021%
6	111	0,0037 %	39	0,030%
7	6	0,0002 %	2	0,002%
8	40	0,0013 %	14	0,011%
9	11	0,0004 %	3	0,002%
> 9	16	0,0005 %	4	0,003%
Total	2 981 061	100,0000 %	128 910	100,000%

TAB. 6.12 – Analyse des données de l'IRISA avec *WebAnalyzer* seul

web de l'IRISA.

Concernant les niveaux de sévérité 1 à 4 on observe les mêmes tendances que pour le serveur web de Supélec. La similarité observée entre les serveurs web de Supélec et de l'IRISA relève du fait que les requêtes suspectes sont en grande partie dues à des procédés automatisés susceptibles de toucher indifféremment n'importe quel serveur accessible sur Internet. En revanche, on observe que le serveur web de l'IRISA héberge un grand nombre de scripts CGI, ce qui constitue une part plus conséquente des alertes émises. Pour les niveaux de sévérité 1 à 4, on dénombre 46 ressources cibles d'attaques.

Les résultats relatifs aux sévérités entre 5 et 9 sont plus mitigés que pour le serveur web de Supélec. En effet, les ressources des niveaux de sévérités 5 et 6 sont pour moitié des sources de fausses alertes. En revanche, on ne trouve qu'une seule ressource engendrant des fausses alertes parmi les niveaux de sévérité entre 7 et 9. Au total, pour les niveaux de sévérité entre 7 et 9, on dénombre 37 ressources cibles d'attaques.

Parmi les quatre ressources engendrant des requêtes de sévérité supérieure ou égale à 10, deux engendrent des fausses alertes.

6.2.3 Un serveur web de France Télécom

Nous présentons ici les résultats d'analyse de sept jours de logs consécutifs du serveur web de France Télécom (du 23 au 31 octobre 2002). Le corpus analysé est composé de 2 725 361 requêtes vers 859 ressources. Les résultats obtenus sont présentés dans le tableau 6.13.

Contrairement aux serveurs web de Supélec et de l'IRISA, on note qu'il n'y a aucune activité ayant engendré des alertes de sévérité supérieure ou égale à cinq. En effet, le serveur web de France Télécom n'est accessible que depuis l'intranet de France Télécom et est donc préservé des attaques de type *scans* que l'on rencontre habituellement sur Internet.

Sévérité	Nb requêtes	Pourcentage	Nb ressources	Pourcentage
0	2 702 615	99,165%	722	84,051%
1	21 739	0,798%	87	10,128%
2	919	0,034%	39	4,540%
3	88	0,003%	6	0,698%
4	20	0,000%	5	0,582%
5	0	0,000%	0	0,000%
6	0	0,000%	0	0,000%
7	0	0,000%	0	0,000%
8	0	0,000%	0	0,000%
9	0	0,000%	0	0,000%
> 9	0	0,000%	0	0,000%
Total	2 725 361	100,000%	859	100,000%

TAB. 6.13 – Analyse des données de France Télécom avec *WebAnalyzer* seul

	Supélec	IRISA	FT
vrais positifs 1-4	104	46	0
faux positifs 1-4	7 105	48 457	137
faux négatifs 1-4	0	0	0
vrais positifs 5-9	12	37	0
faux positifs 5-9	90	47	0
faux négatifs 5-9	0	0	0
vrais positifs > 9	3	2	0
faux positifs > 9	1	2	0
faux négatifs > 9	0	0	0

TAB. 6.14 – Synthèse des résultats relatifs au nombre de faux positifs et vrais positifs avec *WebAnalyzer* seul, en nombre de ressources

Les requêtes n'ayant déclenché aucune alerte sont exclusivement des requêtes légitimes vers des ressources statiques.

Concernant le niveau de sévérité de 1 à 4, on ne recense aucune attaque. En effet, les requêtes ayant activé une ou plusieurs signatures contiennent des encodage non ASCII, utilisent des méthodes HTTP non usuelles (`HEAD`) ou sont relatives à des extensions IIS dont le serveur ne dispose pas.

Conclusion

Les résultats obtenus avec *WebAnalyzer* montrent que la sensibilité de l'outil offre un diagnostic hiérarchisé, permettant à un opérateur de sécurité de se concentrer sur les alertes de plus grande sévérité en premier lieu. De plus l'analyse manuelle des résultats montre que *WebAnalyzer* n'a généré aucun faux négatifs.

En revanche, la sensibilité de cet outil tend à engendrer de trop nombreuses fausses alertes comme l'illustre le tableau 6.14. Ce tableau synthétise les résultats que nous avons obtenus précédemment et énumère, pour chaque serveur web, pour chaque ensemble de sévérité, le nombre de ressources générant des vrais positifs (alertes) et le nombre de ressources générant des faux positifs (fausses alertes). Même si l'analyse effectuée par *WebAnalyzer* n'a donné lieu à aucun faux négatifs, nous avons inscrit les lignes relative dans ce tableau afin de pouvoir comparer ces résultats avec ceux réalisés avec la combinaison $A_u \rightarrow M$ à la section suivante.

Même si on peut considérer que les requêtes ayant un niveau de sévérité allant de 1 à 4 ne représentent pas de danger pour le serveur, il en va tout autrement pour les niveaux de sévérité allant de 5 à 9. En effet, pour ces niveaux de sévérité, *WebAnalyzer* génère un trop grand nombre de fausses alertes pour permettre une réponse rapide et fiable de la part de l'opérateur de sécurité. A l'usage, un opérateur de sécurité aurait tôt fait de ne se préoccuper que des requêtes ayant une sévérité supérieure ou égale à 9 et risquerait ainsi de manquer de nouvelles attaques que l'outil aurait pu partiellement détecter.

6.3 Résultats obtenus avec la combinaison en série $A_u \rightarrow M$

Nous présentons dans cette section les résultats obtenus avec la combinaison en série $A_u \rightarrow M$. Nous appliquons la combinaison sur les serveurs web de Supélec, l'IRISA et France Télécom. Nous utilisons les modèles de comportement établis à la Section 6.1. Les données analysées par la combinaison $A_u \rightarrow M$ sont les mêmes que celles analysées par *WebAnalyzer* seul à la Section 6.2. Les données analysées par la combinaison $A_u \rightarrow M$ sont directement consécutives à celles utilisées pour constituer les modèles de comportement des serveurs de Supélec, de l'IRISA et de France Télécom. Dans ce contexte, les modèles de comportement utilisés ne peuvent être considérés comme obsolètes et les résultats expérimentaux présentés nous donnent une image effective de leur complétude.

Nous présentons, pour chacun des serveurs web, les résultats de l'analyse sous forme de tableau. La première colonne indique les différents niveaux de sévérité. La deuxième colonne indique le nombre de requêtes. La troisième colonne nous montre le pourcentage de requêtes pour chaque niveau de sévérité. La quatrième colonne indique le nombre de ressources différentes vers lesquelles sont effectuées les requêtes. La dernière colonne indique le pourcentage de ressources.

Les résultats présentés sont ceux fournis par *WebAnalyzer* après que *WebFilter* ait filtré les requêtes reconnues comme saines. Nous rappelons ici que l'action de *WebFilter* ne consiste qu'à reconnaître les requêtes vers les ressources qui constituent son modèle de comportement et à les soustraire à l'analyse effectuée par *WebAnalyzer*. Cette action de filtrage ne modifie pas le niveau de sévérité des requêtes qu'il n'a pas su filtrer. Les pourcentages de requêtes (resp. ressources) sont proportion du nombre de requêtes (resp. ressources) initial et non pas proportion du nombre de requêtes (resp. ressources) analysées par *WebAnalyzer* dans la combinaison $A_u \rightarrow M$.

Sévérité	Nb requêtes	Pourcentage	Nb ressources	Pourcentage
0	36349	4,574%	6483	24,773%
1	3369	0,424%	1534	5,862%
2	8078	1,017%	614	2,346%
3	176	0,022%	115	0,439%
4	76	0,010%	14	0,053%
5	11	0,001%	4	0,015%
6	42	0,005%	8	0,031%
7	3	0,000%	3	0,011%
8	0	0,000%	0	0,000%
9	0	0,000%	0	0,000%
> 9	113	0,014%	3	0,011%
Total	48217	6,068%	8778	33,531%

TAB. 6.15 – Analyse des données de Supélec avec la combinaison $A_u \rightarrow M$

Dans ce contexte, le niveau de sévérité 0 correspond aux requêtes et ressources non qualifiées par aucun des deux module de détection composant la combinaison $A_u \rightarrow M$.

6.3.1 Le serveur web de Supélec

Pour l'utilisation de la combinaison $A_u \rightarrow M$ sur les données du serveur de Supélec, nous utilisons le modèle de comportement établi à la Section 4.3.5. Le modèle de comportement est créé à partir des 31 jours de log du mois de mars 2003. Nous appliquons la combinaison $A_u \rightarrow M$ sur les 30 jours suivants (avril 2003). Les résultats sont présentés dans le tableau 6.15.

Pour le serveur de Supélec, on observe que l'application de la combinaison en série $A_u \rightarrow M$ permet de filtrer près de 94% des requêtes. Pour ce faire, *WebFilter* à reconnu seulement 76,5% des ressources demandées.

Sévérité 0

Concernant le niveau de sévérité 0, on observe que *WebFilter* a eu une action très efficace puisqu'il a filtré 590 774 requêtes sur les 627 123 initiales, soit un taux de 94,2%. En revanche, du point de vue des ressources, on observe que pour ce niveau de sévérité, *WebFilter* n'a reconnu que 12 372 ressources, soit un taux de 65,62%. Nous avions déjà analysé que le niveau de sévérité 0 ne contenait aucune attaque dans la Section 6.2.1. Le filtrage effectué n'engendre aucun faux positifs ici.

Nous avons réalisé une analyse manuelle avec les ressources saines non retenues pour faire partie du modèle de comportement pour le niveau de sévérité 0. Nous observons que l'utilisation de ces ressources aurait permis de filtrer des requêtes de sévérité nulle vers 3034 ressources supplémentaires (11,69% de ressources), soit près de la moitié des ressources non identifiées. 16630 requêtes (soit 2,09%) supplémentaires de sévérité nulle auraient ainsi pu être filtrées. Les 3449 ressources restantes n'appartiennent pas

au corpus d'apprentissage utilisé pour créer le modèle de comportement de *WebFilter* et les 19719 requêtes associées n'auraient pu être filtrées.

Sévérité 1 à 4

Pour les niveaux de sévérité de 1 à 4, *WebFilter* a filtré 155 356 requêtes soit 93% des requêtes appartenant initialement à ce niveau de sévérité. Comme pour le niveau de sévérité 0, *WebFilter* n'a ici reconnu que 68,41% de ressources initialement visées, soit 4 932 ressources sur les 7 209 ressources initiales. Les requêtes filtrées sont quasiment toutes dirigées vers des ressources saines. Cependant, on observe aux niveaux de sévérité 1 et 2 le filtrage de certaines requêtes visant des scripts CGI relatifs à l'envoi de *spam*, et le filtrage de requêtes vers une ressource visée par le vers Nimda. On dénombre 86 ressources cibles d'attaques, contre 104 lors de l'analyse effectuée avec *WebAnalyzer* seul, soit 18 ressources générant des faux négatifs. Cependant, les requêtes faites vers ces ressources étaient considérées comme bénignes lors de l'analyse avec *WebAnalyzer* seul puisqu'elles ne mettaient pas en danger le serveur web. Dans le contexte de la combinaison, elles ne sont plus visibles par l'opérateur de sécurité.

Si les ressources saines non retenues dans le modèle de comportement avaient été utilisées elles-aussi, ce sont 9126 requêtes (resp. 672 ressources) supplémentaires qui auraient été filtrées (resp. reconnues), soient 5,46% du total des requêtes (resp. 9,32% des ressources) de sévérité 1 à 4.

Sévérité 5 à 9

Pour les niveaux de sévérité de 5 à 9, *WebFilter* a filtré 81,99% des requêtes, soit 255 requêtes, et 45,01% des ressources, soit 46 ressources. Toutes les requêtes filtrées concernent des demandes de ressources légitimes, sauf une ressource relative à l'activité du ver Nimda (sévérité 5). Cependant, d'autres requêtes issues de ce ver sont présentes parmi les alertes de sévérité 5, son activité n'est donc pas totalement masquée. Les seules fausses alertes restantes concernent les trois requêtes ayant une sévérité de niveau 7 qui contiennent des encodages propres à être utilisés dans une attaques et du code JavaScript. Ces requêtes sont vraisemblablement des erreurs d'édition lors de la création du lien menant vers les ressources légitimes. Pour les niveaux de sévérité 5 à 9, on dénombre 11 ressources cibles d'attaques, contre 12 lors de l'analyse effectuée avec *WebAnalyzer* seul. Nous avons une seule ressource générant des faux négatifs. Cependant, cette ressource n'est qu'une partie de l'activité du ver Nimda qui reste malgré tout decelable.

Sévérité supérieure à 9

Pour le niveau de sévérité supérieur à 9, une seule ressource était initialement source de fausses alertes. Dans la combinaison en série $A_u \rightarrow M$, *WebFilter* l'a filtrée avec succès. Il ne reste effectivement que 3 ressources cibles d'attaques. Il n'y a pas de faux négatifs pour ce niveau de sévérité.

Sévérité	Nb requêtes	Pourcentage	Nb ressources	Pourcentage
0	873 073	29,2873%	53 868	41,7873%
1	342 866	11,5015%	22 661	17,5789%
2	27 844	0,9340%	8 568	6,6465%
3	12 078	0,4052%	2 868	2,2248%
4	3 126	0,1049%	723	0,5609%
5	40	0,0013%	13	0,0101%
6	82	0,0028%	28	0,0217%
7	6	0,0002%	2	0,0016%
8	39	0,0013%	13	0,0101%
9	6	0,0002%	2	0,0016%
> 9	7	0,0002%	3	0,0023%
Total	1 259 167	42,2389%	88 749	68,8457%

TAB. 6.16 – Analyse des données de l'IRISA avec la combinaison $A_u \rightarrow M$

6.3.2 Le serveur web de l'IRISA

Pour l'utilisation de la combinaison $A_u \rightarrow M$ sur les données du serveur de l'IRISA, nous utilisons le modèle de comportement établi à la Section 6.1.1. Le modèle de comportement est créé à partir des 11 derniers jours de log du mois de septembre 2004. Nous appliquons la combinaison $A_u \rightarrow M$ sur les 28 jours suivants (octobre 2004). Les résultats sont présentés dans le tableau 6.16.

L'application de la combinaison en série $A_u \rightarrow M$ sur le serveur de l'IRISA ne nous permet d'obtenir qu'un filtrage d'environ 58%, en reconnaissant moins de 32% des ressources. Ces résultats contrastent avec ceux obtenus pour le serveur web de Supélec. Nous avançons plusieurs hypothèses pour expliquer ce résultat. Tout d'abord, nous avons vu lors de la création du modèle de comportement que le serveur web de l'IRISA comportait trois fois plus de ressources que le serveur web de Supélec. Ensuite, la période d'apprentissage que nous avons utilisée est relativement courte. Finalement, le processus d'édition des ressources (création, suppression, mise à jour) n'est pas dévolu à une seule équipe, mais laissé libre à de nombreuses personnes (professeurs, thésards, ...). Ces trois raisons font que de nombreuses ressources ne sont pas présentes dans le corpus d'apprentissage et *a fortiori* dans le modèle de comportement.

De plus, nous avons vu lors de l'analyse manuelle des groupes formés par l'analyse statistique que les groupes 1.1 et 2.1, qui n'ont pas été retenus, ne contenait que des ressources légitimes. Nous montrons pour les sévérité 0 et 1 à 4, que l'intégration de tels groupes auraient permis de filtrer beaucoup plus de requêtes. Pour le serveur web de l'IRISA, la constitution du modèle de comportement est directement mise en cause pour la qualité du filtrage qu'elle apporte.

Sévérité 0

Le niveau de filtrage observé pour les requêtes de sévérité nulle est relativement

moins bon que celui observé pour le serveur de Supélec. Seulement 1 437 035 requêtes de sévérité nulle (soit 62,21%) ont été filtrées par *WebFilter*. Cela se traduit par 26 451 ressources (soit 32,93%) reconnues par *WebFilter*. Nous avions vu que le modèle de comportement formés pour le serveur web de l'IRISA contenait 21 ressources susceptibles d'être des cibles d'attaques. La présence de ces ressources dans le modèle de comportement masquent 883 requêtes suspectes à l'opérateur.

L'utilisation des ressources saines non retenues dans le modèle de comportement auraient permis de reconnaître 4 252 ressources, soit 5,3% des ressources générant des requêtes de sévérité nulle. Ce nombre peut paraître faible, mais l'inclusion de ces ressources aurait permis de filtrer 241 680 ressources, soit plus d'un quart des requêtes de sévérité nulle n'ayant pas été filtrées (10,46% des requêtes ayant une sévérité nulle).

Sévérité 1 à 4

Pour les niveaux de sévérité de 1 à 4, *WebFilter* a filtré 284 746 requêtes, soit 42,46% des requêtes appartenant initialement à ces niveaux de sévérité. *WebFilter* a reconnu 13 683 ressources, soit 28,21% des ressources visées initialement pour les niveaux de sévérité de 1 à 4. Les 21 ressources intégrées indûment au modèle de comportement filtrent uniquement des requêtes qui auraient engendré des alertes de sévérité de niveaux 1 à 4. La quasi totalité des requêtes ayant un niveau de sévérité compris entre 1 et 4 visent des ressources légitimes. On dénombre 44 ressources cibles d'attaques, contre 46 lors de l'analyse effectuée avec *WebAnalyzer* seul, soit 2 ressources générant des faux négatifs. Ces deux ressources concernent une tentative de scan vers une application mail et une utilisation du serveur web comme proxy.

On observe que l'intégration de 21 ressources illégitimes au modèle de comportement n'engendre le filtrage que de deux ressources cibles d'attaques. Cela s'explique par le fait qu'on ne retrouve pas nécessairement les ressources composant le corpus d'apprentissage dans les ressources que nous analysons avec la combinaison en série $A_u \rightarrow M$.

L'utilisation des ressources saines non incluses dans le modèle comportemental aurait permis de reconnaître 1 226 ressources supplémentaires, soit 2,52% des ressources présentes initialement dans les niveaux de sévérité 1 à 4. Cependant, ces ressources génèrent à elles seules 179 174 requêtes, ce qui signifie que l'intégration de ces ressources au modèle de comportement de *WebFilter* auraient permis de filtrer 46,43% des requêtes de sévérité 1 à 4.

Sévérité 5 à 9

Pour les niveaux de sévérité de 5 à 9, *WebFilter* a reconnu 26 ressources, soit 30,95% des ressources de ces niveaux de sévérité. Au niveau des requêtes, cela se traduit par le filtrage de 104 requêtes, soit 37,54% des requêtes de ces niveaux de sévérité. Les ressources générant des faux positifs sont aux nombre de 21, cantonnées aux niveaux de sévérité 5 et 6. Le filtrage de *WebFilter* n'occasionne aucun faux négatif pour ces niveaux de sévérité. Pour les niveaux de sévérité 5 à 9, on dénombre 37 ressources cibles d'attaques, comme lors de l'analyse effectuée avec *WebAnalyzer* seul.

Sévérité	Nb requêtes	Pourcentage	Nb ressources	Pourcentage
0	162 142	5,9493%	229	26,6589%
1	3 392	0,1244%	32	03,7253%
2	113	0,0041%	32	03,7253%
3	38	0,0014%	3	00,3492%
4	20	0,0007%	5	00,5821%
5	0	0,0000%	0	00,0000%
6	0	0,0000%	0	00,0000%
7	0	0,0000%	0	00,0000%
8	0	0,0000%	0	00,0000%
9	0	0,0000%	0	00,0000%
> 9	0	0,0000%	0	00,0000%
Total	165 705	6,0800%	301	35,0407%

TAB. 6.17 – Analyse des données de France Télécom avec la combinaison $A_u \rightarrow M$

Sévérité supérieure à 9

Concernant le niveau de sévérité supérieur 9, *WebFilter* n'a reconnu qu'une seule des deux ressource générant des faux positifs. On dénombre donc, pour ce niveau de sévérité, 2 ressources cibles d'attaques (comme pour l'analyse effectuée avec *WebAnalyzer* seul) et une ressource générant des fausses alertes.

6.3.3 Un serveur web de France Télécom

Pour l'utilisation de la combinaison $A_u \rightarrow M$ sur les données du serveur de France Télécom, nous utilisons le modèle de comportement établi à la Section 6.1.2. Le modèle de comportement est créé à partir de 7 jours de log, du 16 au 22 octobre 2002 inclus. Nous appliquons la combinaison $A_u \rightarrow M$ sur les 8 jours suivants, du 23 au 31 octobre inclus. Les résultats sont présentés dans le tableau 6.17.

Pour le serveur de France Télécom, on observe que l'application de la combinaison en série $A_u \rightarrow M$ permet de filtrer près de 94% des requêtes grâce à la reconnaissance d'environ 65% des ressources. Ces résultats sont très similaires à ceux observés pour le serveur de Supélec. Ici, contrairement au serveur de l'IRISA, la courte période d'apprentissage ne semble pas être handicapante pour la constitution d'un modèle de comportement efficace. Ceci est sans doute dû au mode d'édition (ajout, suppression, mise à jour) du site web de France Télécom. Une équipe étant chargé de gérer l'édition, on retrouve une grande stabilité dans les ressources présentes sur le serveur. web de Supélec et de l'IRISA.

Sévérité 0

Pour le niveau de sévérité 0, *WebFilter* a une action de filtrage comparable à celle observée sur le serveur web de Supélec. 2 540 473 requêtes de sévérité 0 ont été filtrées sur les 2 702 615 initiales, soit 94% des requêtes de ce niveau de sévérité. En revanche,

	Supélec	**IRISA**	**FT**
vrais positifs 1-4	86	44	0
faux positifs 1-4	2 191	34 776	72
faux négatifs 1-4	18	2	0
vrais positifs 5-9	11	37	0
faux positifs 5-9	4	21	0
faux négatifs 5-9	1	0	0
vrais positifs > 9	3	2	0
faux positifs > 9	0	1	0
faux négatifs > 9	0	0	0

TAB. 6.18 – Synthèse des résultats relatifs au nombre de faux positifs, vrais positifs et faux négatifs avec la combinaison $A_u \rightarrow M$, en nombre de ressources

comme pour le serveur de Supélec, on remarque que ces requêtes filtrées ne concernent que 493 ressources, soit 68,28% des ressources générant des requêtes de sévérité nulle. Le modèle de comportement est exempt de ressources susceptibles d'être des cibles d'attaques. La combinaison en série $A_u \rightarrow M$ n'engendre aucun faux négatifs.

L'intégration des ressources saines exclues du modèle de comportement auraient permis de reconnaître 84 ressources supplémentaires, soit 11,63% de ressources supplémentaires reconnues pour ce niveau de sévérité. 3949 requêtes supplémentaires auraient pu être ainsi filtrées, ce qui ne représente que 0,15% des requêtes pour ce niveau de sévérité.

Sévérité 1 à 4

Pour les niveaux de sévérité de 1 à 4, *WebFilter* a filtré 19 203 requêtes, soit 84,35% des requêtes appartenant initialement à ces niveaux de sévérité. Seules 65 ressources de ces niveaux de sévérité ont été reconnues par *WebFilter*, ce qui correspond à 47,45% des ressources initiales.

L'utilisation des ressources saines exclues du modèle de comportement auraient permis de reconnaître 9 ressources supplémentaires, soit 6,57% des ressources de ces niveaux de sévérité. Ces 9 ressources sont visées par 572 requêtes, ce qui correspond seulement à 2,51% des requêtes de ces niveaux de sévérité.

Conclusion

Dans cette section, nous avons appliqué la combinaison en série $A_u \rightarrow M$ sur les serveurs web de Supélec, l'IRISA et France Télécom. Les résultats obtenus sont très encourageants pour la réduction des faux positifs pour les serveurs de Supélec et de France Télécom, alors qu'ils sont plus mitigés pour le serveur de l'IRISA.

Le tableau 6.18 synthétise les résultats obtenus avec la combinaison $A_u \rightarrow M$ et énumère, pour chaque serveur web, pour chaque ensemble de sévérité, le nombre de ressources générant des vrais positifs (alertes), le nombre de ressources générant des faux positifs (fausses alertes) et le nombre de ressources générant des faux négatifs

(attaques non signalées).

Ces résultats montrent une réelle action concernant le problème des faux positifs en détection d'intrusions, notamment pour les niveau de sévérité supérieurs à 5, qui représentent des attaques caractérisées auxquelles le serveur est potentiellement vulnérable.

On remarque aussi que pour les niveaux de sévérité supérieurs à 5, l'introduction de faux négatifs par le filtrage de *WebFilter* se limite à une seule ressource.

Sur les expériences que nous avons menées, nous observons donc que l'utilisation de la combinaison $A_u \rightarrow M$ permet de réduire de façon significative le nombre de faux positifs, comparé à une utilisation seule de *WebAnalyzer*, tout en limitant le risque de faux négatifs. De plus, concernant les performances, l'action de filtrage réalisée par *WebFilter*, peu complexe en temps, soulage nettement la charge d'analyse de *WebAnalyzer* qui est elle beaucoup plus grande.

6.4 Conclusion

Dans ce chapitre, nous avons réalisé des expérimentations relatives à trois aspects. Tout d'abord, la construction de modèle de comportement suivant l'analyse statistique présentée au Chapitre 4, aidée des profils que nous avions établis. Ensuite, l'analyse effectuée par *WebAnalyzer* seul montrant ses capacités de diagnostic. Finalement, la mise en œuvre de la combinaison $A_u \rightarrow M$, utilisant les modèles de comportement créés, illustrant les capacités de filtrage de *WebFilter* et plus particulièrement la réduction du nombre de faux positifs.

Les résultats des analyses effectuées avec *WebAnalyzer* seul nous montre la capacité de diagnostic de cet outil. En effet, la hiérarchisation apportée par les différents niveaux de sévérité permet à l'opérateur de se consacrer en premier lieu aux attaques les plus sérieuses. De plus, la sensibilité de *WebAnalyzer* a permis de détecter toutes les attaques présentes. Cependant, cette sensibilité a une contre-partie importante qui consiste en l'émission de trop nombreuses fausses alertes. Il pourrait être difficile pour un opérateur de sécurité d'utiliser cet outil, à moins de ne considérer que les niveaux de sévérité les plus élevés. L'utilisation de la combinaison $A_u \rightarrow M$ permet justement de filtrer en amont les ressources reconnues comme saines, filtrant ainsi *de facto* un grand nombre de fausses alertes.

Les résultats expérimentaux montrent que l'automatisation apportée par l'utilisation des profils établis au Chapitre 4 permet de créer des modèles de comportement relativement efficaces. En effet, *WebFilter* a pu filtrer entre 57% et 94% des requêtes soumises à un serveur web. De plus, cette automatisation n'engendre pas pour autant de biais important dans l'analyse des attaques contre le serveur web. En effet, sur les trois serveurs web la quasi totalité des faux négatifs sont engendrés pour des sévérité de niveau 1 à 4, niveaux de sévérité qui représente les attaques bénignes telles que les tentatives de scans par exemple. Une seule ressource générant des faux négatifs pour les niveaux de sévérité 5 à 9 est à déplorer.

En revanche, le faible taux de filtrage observé pour le serveur web de l'IRISA montre les limites de notre approche quand à la découverte de nouvelles attaques. En effet,

les requêtes de sévérité nulle en sortie de la combinaison $A_u \rightarrow M$ est l'ensemble des requêtes n'ayant été qualifiées par aucun des deux composants. Si cet ensemble de requêtes est trop important, un opérateur de sécurité s'en désintéressera et les considèrera, à l'usage, comme des requêtes légitimes. Il semble donc qu'il faille adapter le processus de création du modèle de comportement pour ce site web, notamment le très grand nombre de ressources que nous avons eu à traiter. De même, la durée d'apprentissage, ainsi que la stabilité des ressources présentes sur le serveur sont deux caractéristiques importantes dont il faut tenir compte.

Plus généralement, la présence de requêtes vers des ressources n'appartenant pas au corpus d'apprentissage pose la question classique de l'approche comportementale : la mise à jour du modèle de comportement. Nous n'avons pas abordé cette problématique dans nos travaux. Cependant, la structure du modèle de comportement, basée sur la structure des ressources, permet à un opérateur de sécurité de n'avoir qu'à analyser les ressources visées qui sont en moindre nombre que les requêtes. Ainsi, un simple ajout des ressources déclarées saines après analyse de l'opérateur pourrait constituer une mise à jour suffisante à court terme.

Chapitre 7

Conclusion

Les outils de détection d'intrusions constituent une seconde ligne de défense, complétant l'action des outils de sécurité préventifs tels que les *firewalls* ou proxys. Cependant, les outils de détection d'intrusions sont connus pour émettre de nombreuses fausses alertes, et les alertes émises se révèlent assez pauvre en information sur l'attaque qui les a engendrées.

Nous avons choisi d'étudier une approche utilisant les deux méthodes de détection, comportementale et par scénarios, afin d'utiliser les capacités de qualification de chacune. Nous appliquons cette approche sur des logs applicatifs. Cette source de données nous permet de concentrer les efforts de notre combinaison sur un service spécifique. En effet, les sondes réseaux visant à surveiller toute l'activité ont des contraintes de performances très forte qui les oblige à utiliser des signatures de détection génériques, moins précises. Ce manque de précision est une des raison de l'émission de fausses alertes. De plus, l'utilisation de logs applicatifs nous permet de traiter une information de haut niveau qui, même si elle n'est pas complète, décrit de façon synthétique les interactions entre le service surveillé et les utilisateurs.

Dans ce dernier chapitre, nous résumons nos contributions et dessinons les perspectives qu'amènent nos travaux.

7.1 Contributions

Nous avons présenté dans le Chapitre 3 une formalisation du diagnostic des méthodes de détection d'intrusions comportementale et par scénarios en fonction de la vraie nature des événements. Ce formalisme nous permet de raisonner sur l'opportunité de combiner des méthodes détection d'intrusions. Le but est d'utiliser la combinaison de méthodes de détection d'intrusions pour améliorer le diagnostic en éliminant le plus de faux positifs, chaque méthode tant utilisée pour qualifier au mieux les événements analysés.

Le formalisme que nous avons établi permet d'évaluer la pertinence des combinaisons de méthodes de détection d'intrusions en fonction de problématiques et de contextes spécifiques. Ce formalisme et la nature des événements que nous voulons analyser nous

a conduit à utiliser la combinaison $A_u \rightarrow M$. En effet, la reconnaissance des événements sains effectuée en amont par la détection d'intrusions comportementale permet potentiellement de réduire le nombre de fausses alertes émises finalement par la méthode de détection d'intrusions par scénarios. De plus, grâce à cette combinaison, la capacité de qualification de chacune des deux méthodes de détection d'intrusions est utilisée pleinement. Les événements à traiter étant très majoritairement sains, le module de détection d'intrusions comportementale est chargé de reconnaître en premier lieu. Les événements non qualifiés sont ensuite analysés par le module de détection d'intrusions par scénarios. Avec cette combinaison, l'opérateur de sécurité dispose d'un diagnostic à trois états. Les événements sont reconnus soit sains, soit intrusifs, soit non qualifiés, contrairement à l'utilisation usuelle des outils de détection d'intrusions tels que Snort où tout événement qui n'est pas reconnu comme une attaque est considéré comme sain.

Notre seconde contribution consiste en la mise en œuvre d'un outil de détection d'intrusions comportementale destiné à être utilisé dans la combinaison en série $A_u \rightarrow M$, présenté au Chapitre 4 : *WebFilter*. Tout d'abord, la phase de détection de *WebFilter* est peu coûteuse comparé à la recherche de plusieurs motifs sous forme de *pattern matching* telle qu'on la retrouve dans les outils de détection d'intrusions par scénarios par exemple. La structure arborescente limite la complexité de la reconnaissance d'un événement à la profondeur de l'arbre qui constitue le modèle de comportement.

Ensuite, la construction du modèle de comportement et les résultats obtenus montrent que l'utilisation de méthodes statistiques connues et éprouvées permet d'établir des profils de comportement. Contrairement à la détection d'intrusions comportementale basée sur des mécanismes de seuils [15, 42], seuils définissant les limites acceptables d'un comportement, les profils établis peuvent servir de référence pour la construction de modèles de comportement relatifs à d'autres serveurs web. On retrouve cette notion de profils chez Julisch et *al.* [35], mais appliquée à des alertes émises, non pas à des événements non qualifiés.

L'automatisation apportée par l'emploi de méthodes statistiques et l'utilisation des profils permet un gain de temps substantiel et nécessite une expertise moindre de la part de l'opérateur de sécurité, comparé à une analyse manuelle.

Dans le cadre de la combinaison $A_u \rightarrow M$, *WebFilter* permet de filtrer de 54% à 94% des événements. De plus, ce procédé de création de modèle de comportement n'engendre selon nos expérimentations que quelques faux négatifs, qui se révèlent bénins après analyse.

La contribution apportée par *WebAnalyzer* est la précision du diagnostic qu'il apporte sur les événements intrusifs visant un serveur web. La précision du diagnostic est obtenue en utilisant la combinaison des signatures ayant été activée par une requête, permettant ainsi de décrire toutes les composantes intrusives constituant cette dernière. Ce mode de fonctionnement s'oppose à celui d'autres IDS, tel que Snort [70], qui stoppent l'analyse d'un événement dès qu'une signature d'attaque est activée. En effet, par souci de performance, l'analyse s'arrête au plus tôt afin d'éviter d'appliquer des signatures inutilement. *WebAnalyzer* est moins soumis a ces contraintes de performance. Il analyse les données d'audit applicatives, donc des données plus synthétiques que des données réseau par exemple, et ne provenant de l'activité que d'une seule application.

Les signatures utilisées par *WebAnalyzer* visent à reconnaître les applications pour lesquelles une vulnérabilité est connue, mais aussi les caractéristiques communément utilisée lors de l'exploitation d'une vulnérabilité. Ces caractéristiques sont, par exemple, des encodages non ASCII qui peuvent révéler une tentative de *buffer overflow*. *WebAnalyzer* recherche aussi si des éléments sensibles du systèmes sont visés, tels que les tentatives d'accès à des données systèmes ou l'activation de commandes. Même si nous ne l'avons pas observé dans nos expérimentation, ces caractéristiques permettent de détecter des attaques inconnues, utilisant ces méthodes.

L'utilisation des codes de retour (*status code*) émis par le serveur web permet de déterminer la portée d'une éventuelle attaque. La description précise d'une attaque, ainsi que la connaissance de sa portée, permettent de lui associer un niveau de sévérité. L'opérateur dispose alors d'un diagnostic hiérarchisé, lui permettant de prendre les contre-mesures nécessaires aux attaques les plus dangereuses, chaque attaque étant complètement décrite grâce à la combinaison de signatures.

Les travaux relatifs à *WebAnalyzer* ont été publiés à la conférence EICAR [13] et à la conférence SAR [14], en 2005. Une version de cette article sera publié dans les Annales des Télécommunications. L'ensemble de la combinaison $A_u \rightarrow M$, le formalisme ainsi que des expérimentations ont été publiés à la conférence ACSAC 2004 [75].

7.2 Perspectives

Les travaux présentés dans cette thèse ouvrent tout d'abord des perspectives à court terme, principalement relatives à des améliorations de *WebFilter* et *WebAnalyzer*.

Concernant *WebFilter*, l'amélioration que nous souhaitons apporter concerne la prise en compte de la valeur des paramètres et de leur corrélation lors de la phase de détection. Cette analyse de la valeur des paramètres permettrait de vérifier leur validité propre, mais aussi en fonction de la valeur des autres paramètres utilisés. Un brevet initiant des travaux dans ce sens a été déposé [74].

L'étude d'autres algorithmes de *clustering* offre aussi des perspectives afin d'améliorer la qualité des groupes formés et donc la qualité du modèle de comportement. Elle permettrait éventuellement de pallier le problème du trop grand nombre de ressources que nous avons rencontré avec le serveur web de l'IRISA.

A court terme, pour *WebAnalyzer*, nous souhaitons pouvoir adapter le fichier de signatures utilisé en fonction du type de serveur analysé, de l'organisation des ressources ainsi que des applications qu'il héberge. Ceci pourrait être fait soit par apprentissage automatique, via les fichiers de configuration du serveur web par exemple, soit par configuration de l'outil en fonction d'une politique de sécurité.

A plus long terme, nous souhaitons travailler à l'amélioration du modèle de comportement de *WebFilter*. Le premier concerne l'étude des besoins en terme de quantité de données dans le corpus d'apprentissage pour obtenir un modèle de comportement le plus complet possible. En effet, les résultats mitigés que nous obtenons pour le serveur web de l'IRISA sont certainement dus à une période d'apprentissage mal adaptée ainsi qu'à une grande instabilité des ressources hébergées sur le serveur web. Actuellement,

nous ne savons pas quantifier, pour un serveur web donné, la durée d'apprentissage nécessaire qui serait représentative de son activité.

Le second point concerne la maintenance du modèle de comportement. En effet, le contenu d'un serveur web est amené à évoluer, ainsi que le comportement de ses utilisateurs. Il faut donc être capable, au cours du temps, d'intégrer ou de supprimer des ressources du modèle de comportement. Il semble que l'utilisation des profils que nous avons établis puissent être utilisés, tant que ceux-ci sont représentatif des comportements des utilisateurs. Il paraît nécessaire d'étudier les options de maintenance possibles, qu'elles s'effectuent en continu ou de façon ponctuelle.

Nous souhaitons étudier et évaluer toutes les combinaisons de méthodes de détection d'intrusions afin de déterminer les éventuelles solutions qu'elles peuvent apporter en fonction de contextes spécifiques. Nous avons présenté dans cette thèse les combinaisons $A_u \rightarrow M$ et $M_i \rightarrow A$. Nous avons étudié et mis en œuvre la combinaison $A_u \rightarrow M$ car elle correspond à nos besoins de qualification des événements dans un contexte où la majorité de ces événements sont sains. Les combinaisons $A_s \rightarrow M$ et $M_u \rightarrow A$ sont hors de propos concernant notre problématique. Cependant, la combinaison $M_u \rightarrow A$ peut être utilisée dans un contexte où le composant de détection d'intrusions par scénarios émet peu de faux positifs. Le composant de détection d'intrusions par scénarios sert alors à détecter d'éventuelles nouvelles attaques qui n'auraient pas pu être identifiées en premier lieu.

Finalement, nous souhaitons appliquer la combinaison $A_u \rightarrow M$ à une analyse multi-événementielle. Actuellement, nous sommes capables de détecter des attaques directes contre un serveur web, c'est-à-dire qu'une attaque correspond à une seule requête dans laquelle est exploitée une vulnérabilité. En revanche, nous sommes incapables de déterminer, par exemple, si un utilisateur s'est authentifié avant d'obtenir une ressource dont l'accès est restreint. A terme, l'objectif est de pouvoir généraliser l'utilisation de la combinaison $A_u \rightarrow M$ à tous types de données, permettant à la méthode de détection d'intrusions comportementale de filtrer en amont les événements sains, avant que les événements non reconnus ne soient analysés par la méthode de détection d'intrusions par scénarios.

Annexe A

Données complètes

A.1 Supélec

	Groupe 1		Groupe 2		Groupe 3	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,09031	0,749363780	0,2580	1,956512e-01	0,3136	0,09897138
Req-Jour	0,15769	0,004697373	1,80660	1,357322e+00	1,09149	0,14205287
PctReq-Jour	5,119e-06	0,005023764	6,473e-05	1,365911e+00	3,939e-05	0,14828811
PctUserId	0,0	0,0	0,0024	1,394773e+00	0,001052	0,15942283
PctGet	0,3978	10,418137209	0,9965	4,573287e-02	0,9752	0,39601834
PctPost	0,1628	79,564295611	1,492e-05	1,650390e-03	0,001879	0,54235372
PctHead	0,3836	22,960410670	0,003324	9,222777e-02	0,02157	0,24169434
PctVide	0,02326	98,920456278	0,0	3,009502e-36	0,00000000	0,00000000
PctAutresMeth	0,03256	33,496034864	0,0001545	3,026314e-02	0,001316	0,69795234
PctHTTP/1.x	0,9767	81,657386372	1,0	1,203801e-35	0,9994	1,70699591
Pct200	0,1375	0,509252259	0,9751	1,608807e-02	0,005521	0,01450138
Pct300	0,002016	0,032171653	0,02336	1,669448e-01	0,0009196	0,01410233
Pct400	0,8372	0,576753272	0,0009603	8,592268e-04	0,9931	0,01884638
Pct500	0,02326	35,343417322	0,000549	8,498081e-01	0,0004513	0,69887856
args	0,1721	13,748246269	0,01121	1,065137e+00	0,01354	1,28323364

	Groupe 4		Groupe 5		Groupe 6	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,004165	9,908145e-03	0,990	4,587510e-02	0,2330	1,286984745
Req-Jour	0,07206	2,709220e-03	0,03894	3,064173e-05	5,34105	9,809445035
PctReq-Jour	2,393e-06	2,657695e-03	1,811e-06	4,654393e-05	1,880e-04	9,481776500
PctUserId	0,000698	3,102732e-01	0,0006484	2,059050e-01	0,007772	4,655090217
PctGet	1,000	7,890662e-04	1,0	0,000000e+00	0,9472	5,911988492
PctPost	0,0	0,000000e+00	0,0	0,000000e+00	0,001299	0,378690394
PctHead	8,599e-06	8,703185e-05	0,0	0,000000e+00	0,01193	0,407598190
PctVide	0,0	3,009938e-36	0,0	0,000000e+00	0,0	0,000000000
PctAutresMeth	2,885e-05	3,939871e-03	0,0	0,000000e+00	0,03955	30,39519627
PctHTTP/1.x	1,0	0,000000e+00	1,0	0,000000e+00	1,0	0,000000000
Pct200	0,0007119	7,600042e-04	0,0004828	5,145677e-04	0,05213	0,087732012
Pct300	0,0003036	6,155138e-03	0,0	0,000000e+00	0,9465	1,166714377
Pct400	0,9988	2,300942e-03	0,9995	5,009554e-04	0,001335	0,001002924
Pct500	0,0002236	3,462363e-01	0,0	0,000000e+00	0,0	0,000000000
args	0,002459	2,356673e-01	0,003071	2,942554e-01	0,005181	0,496353394

TAB. A.1 – Caractéristiques des 6 groupes résultants d'un partitionnement utilisant la méthode CLARA. Les variances sont calculées sur les variables centrées et réduites

A.2 IRISA

A.2.0.1 1ère partie – 9 groupes

	Groupe 1		Groupe 2		Groupe 3		Groupe 4	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,02140	5,687101e-02	0,19139	2,635432e-01	0,1807	1,05097297	0,1515	1,0808035483
Req-Jour	0,36982	1,923656e-02	2,5181	3,231980e+00	0,59877	0,25667952	0,37108	0,0245552269
PctReq-Jour	4,130e-06	2,147505e-02	2,866e-05	3,201836e+00	6,891e-06	0,24887548	4,290e-06	0,0268643635
PctUserId	0,0007219	3,333808e-01	0,0004068	2,743716e-01	0,0	0,00000000	0,0	0,0000000000
PctGet	0,9973	6,845789e-01	0,9967	3,757518e-01	0,9997	0,00924342	0,9952	1,0676293400
PctPost	0,0	0,000000e+00	6,494e-05	1,598832e-02	0,0003274	0,01797998	0,0	0,0000000000
PctHead	0,002206	2,034236e+00	0,002217	6,504133e-01	0,0	0,0	0,002177	1,4383483208
PctVide	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0000000000
PctAutresMeth	0,0005183	6,014253e-01	0,001028	8,787428e-01	0,0	0,0	0,002669	3,1238329066
PctHTTP/1.x	1,0	0,000000e+00	1,0000	8,966343e-03	0,9998	16,11363786	1,0000	1,6542897063
Pct200	0,0005871	8,273369e-04	0,8588	1,052191e-01	0,9336	0,27341009	0,01356	0,0204320986
Pct300	2,255e-05	7,029016e-06	0,1410	1,815885e-01	0,004103	0,01359376	0,9863	0,0355654216
Pct400	0,9994	1,227787e-03	0,00022	4,055410e-04	0,02483	0,17342635	0,0001045	0,0001869151
Pct500	0,0	0,0	3,947e-08	6,503426e-09	0,03728	18,73893455	0,0	0,0000000000
args	0,0	0,0	0,0001525	2,060016e-03	1,0	0,0	0,0	0,0000000000

	Groupe 5		Groupe 6		Groupe 7		Groupe 8		Groupe 9	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,9616	0,1300397376	0,003430	3,792817e-03	0,902	0,445755134	0,1804	1,122183489	0,04748	0,3476064
Req-Jour	0,13464	0,0002040399	0,27115	2,888288e-03	0,15046	0,004186411	0,61778	1,127594454	1,99091	0,4171882
PctReq-Jour	2,072e-06	0,0003147781	2,909e-06	2,658934e-03	2,249e-06	0,006801686	7,722e-06	1,437191769	2,229e-05	0,3998119
PctUserId	0,0	0,0000000000	0,003838	2,789010e+00	0,0	0,000000000	0,0	0,000000000	0,0	0,0000000
PctGet	0,9978	0,4359465976	0,9967	4,941518e-01	0,9993	0,103384790	1,0	0,000000000	0,01	1,1893394
PctPost	0,0	0,0000000000	0,0	4,815497e-35	0,0	0,000000000	0,0	0,000000000	0,99	2,3134609
PctHead	0,002231	1,5991353532	0,002308	9,546914e-01	0,0006954	0,379235148	0,0	0,000000000	0,0	0,0000000
PctVide	0,0	0,0000000000	0,0	0,0000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000
PctAutresMeth	0,0	0,0000000000	0,0009625	1,082359e+00	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000
PctHTTP/1.x	1,0	0,0000000000	1,0	0,0000000000	1,0	0,000000000	1,0	0,000000000	1,0	0,000000000
Pct200	0,977	0,0446784415	0,9952	3,479311e-03	0,0	0,000000000	0,002194	0,005270190	0,4 630	1,3150858
Pct300	0,02302	0,0773142003	0,00435	4,648833e-03	0,0	0,000000000	0,9978	0,009119848	0,0 370	0,1967213
Pct400	0,0	0,0000000000	0,0004181	1,101495e-03	1,0	0,000000000	0,0	0,000000000	0,5	1,8350270
Pct500	0,0	0,0000000000	0,0	0,000000e+00	0,0	0,000000000	0,0	0,000000000	0,0	0,0000000
args	0,0	0,0000000000	0,0	0,000000e+00	0,006954	0,093395165	1,0	0,000000000	1,0	0,0000000

TAB. A.2 – Caractéristiques des 9 groupes résultants d'un partitionnement utilisant la méthode CLARA sur une première partie des données provenant du serveur de l'IRISA. Les variances sont calculées sur les variables centrées et réduites.

A.2.0.2 2ème partie – 9 groupes

	Groupe 1		Groupe 2		Groupe 3		Groupe 4	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,1463	1,343961227	0,1526	1,184916182	0,5085	0,120708197	0,9992	2,289311e-03
Req-Jour	0,51250	0,006787787	1,18135	0,188178453	0,6579	0,007167795	0,11090	1,012246e-05
PctReq-Jour	5,777e-06	0,006582379	1,333e-05	0,188028320	8,634e-06	0,010112596	1,754e-06	1,883488e-05
PctUserId	0,002217	0,086412895	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000
PctGet	0,9935	3,259561539	0,9951	1,534014278	0,9919	1,544948771	0,9987	6,589381e-01
PctPost	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000
PctHead	0,002842	2,312979494	0,003824	2,423736980	0,00806	2,870270226	0,001313	1,224203e+00
PctVide	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000
PctAutresMeth	0,003613	5,648136775	0,001045	0,624328058	0,0	0,000000000	0,0	0,000000000
PctHTTP/1.x	1,0	0,000000000	1,0	0,000000000	1,0	0,000000000	1,0	0,000000000
Pct200	0,001688	0,002585299	0,05289	0,070216889	0,9459	0,065865415	0,9928	2,183211e-02
Pct300	0,0006002	0,002209436	0,9467	0,135788376	0,05326	0,121670068	0,007232	4,168894e-02
Pct400	0,9977	0,006199992	0,0004054	0,001161064	0,0008108	0,003574700	0,0	0,000000000
Pct500	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000
args	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000

	Groupe 5		Groupe 6		Groupe 7		Groupe 8		Groupe 9	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,006413	6,098601e-03	0,20681	8,042341e-02	0,01222	0,14735356	0,1330	0,252492829	0,1892	1,218728888
Req-Jour	0,59362	5,013147e-03	1,8925	1,507932e-01	1,05651	0,01764833	7,6804	6,001350076	0,49232	0,002176865
PctReq-Jour	6,423e-06	4,857489e-03	2,176e-05	1,551098e-01	1,094e-05	0,01444792	8,516e-05	5,996482221	5,775e-06	0,002653507
PctUserId	0,0	1,926177e-34	0,0	0,000000000	0,9615	1,15710854	0,0005736	0,030362423	0,003232	0,203297919
PctGet	0,9983	4,431187e-01	0,9975	3,910491e-01	0,9892	4,06405596	0,9975	0,215722751	0,9917	4,787789781
PctPost	0,0	3,009651e-36	0,0	0,000000000	0,004505	24,97409971	0,0	0,000000000	0,00831	45,818133629
PctHead	0,001304	4,958755e-01	0,001970	4,009920e-01	0,00288	1,98182712	0,002184	0,303627860	0,0	0,000000000
PctVide	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000
PctAutresMeth	0,000418	4,928835e-01	0,0004942	4,914883e-01	0,003396	1,20385579	0,0002832	0,138305220	0,0	0,000000000
PctHTTP/1.x	1,0	3,009651e-36	1,0	0,000000000	1,0	0,00000000	1,0	0,000000000	0,9945	61,255962109
Pct200	0,9905	6,002125e-03	0,9674	1,866038e-02	0,8388	0,62398969	0,6773	0,117934643	0,6124	1,596025642
Pct300	0,009523	1,145784e-02	0,03253	3,559401e-02	0,06259	0,41333244	0,3216	0,226916676	0,3213	2,828664601
Pct400	7,119e-06	3,798175e-06	5,856e-05	4,069448e-05	0,09863	0,61706113	0,001097	0,004853464	0,05875	0,513683606
Pct500	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000
args	0,0	0,000000000	0,0	0,000000000	0,01351	0,82933802	0,0	0,000000000	1,0	0,000000000

Tab. A.3 – Caractéristiques des 9 groupes résultants d'un partitionnement utilisant la méthode CLARA sur une deuxième partie des données provenant du serveur de l'IRISA. Les variances sont calculées sur les variables centrées et réduites.

A.2.0.3 3ème partie – 8 groupes

	Groupe 1		Groupe 2		Groupe 3		Groupe 4	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,02595	0,067482828	0,1774	1,149808933	0,006852	6,756224e-03	0,9461	1,556774e-01
Req-Jour	0,35429	0,007738511	1,20297	0,253748503	0,44107	5,764700e-03	0,14980	2,560239e-04
PctReq-Jour	3,991e-06	0,007227132	1,375e-05	0,224904994	4,747e-06	4,875672e-03	2,272e-06	3,548265e-04
PctUserId	0,0004431	0,451714584	0,0	0,000000000	6,214e-05	7,884279e-02	0,0	0,000000e+00
PctGet	0,9925	0,678253172	0,9968	0,295945651	0,9957	2,811863e-01	0,968	3,276829e+00
PctPost	0,0	0,000000000	0,0	0,000000000	0,0	1,926127e-34	0,0	0,000000e+00
PctHead	0,003418	0,792800395	0,0009583	0,199959079	0,002844	4,565022e-01	0,03055	1,017357e+01
PctVide	0,0	0,000000000	0,0	0,000000000	0,001021	1,774314e+00	0,001414	2,458710e+00
PctAutresMeth	0,004076	4,971292501	0,002273	2,671099786	0,0004206	3,614418e-01	0,0	0,000000e+00
PctHTTP/1.x	0,9986	1,918861672	1,0	0,000000000	0,999	1,441822e+00	0,9986	1,997967e+00
Pct200	0,003159	0,005760854	0,05546	0,094586846	0,9791	3,171493e-02	0,9857	4,270538e-02
Pct300	0,0004968	0,001136190	0,9440	0,189624717	0,01908	4,967567e-02	0,01357	7,615663e-02
Pct400	0,9961	0,011736375	0,0005876	0,001212218	0,001039	3,673831e-03	0,0	0,000000e+00
Pct500	0,0002264	0,309589521	0,0	0,001212218	0,0007403	1,868934e+00	0,0007072	1,933730e+00
args	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000	0,0	0,000000000

	Groupe 5		Groupe 6		Groupe 7		Groupe 8	
Variables	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**	**Moyenne**	**Variance**
PctFérié	0,2761	2,716475e-01	0,1516	0,8531460	0,9564	2,115656e-01	0,00766	0,02732642
Req-Jour	2,4830	3,594813e+00	1,37522	0,7675384	0,10958	5,082724e-05	1,27966	0,04023730
PctReq-Jour	2,866e-05	3,608247e+00	1,581e-05	0,7479678	1,680e-06	6,687895e-05	1,359e-05	0,03171193
PctUserId	0,001675	3,486655e+00	0,0	0,0000000	0,0	0,000000e+00	0,0	0,00000000
PctGet	0,9985	3,809355e-02	0,9946	0,3323900	0,9937	6,714740e-01	0,01385	0,43418080
PctPost	0,0	0,000000e+00	0,003411	0,2741738	0,0	0,000000e+00	0,9861	0,78976698
PctHead	0,001362	9,172427e-02	0,001969	0,5943979	0,001567	5,475047e-01	0,0	0,00000000
PctVide	0,0	1,203908e-35	0,0	0,0000000	0,001567	2,726550e+00	0,0	0,00000000
PctAutresMeth	0,000187	1,125010e-01	0,0	0,0000000	0,003135	3,831157e+00	0,0	0,00000000
PctHTTP/1.x	1,0	4,312978e-07	1,0	0,0000000	0,9984	2,215616e+00	1,0	0,00000000
Pct200	0,8763	1,705417e-01	0,6621	1,4111637	0,0	0,000000e+00	0,8724	0,62532125
Pct300	0,1232	3,375238e-01	0,2963	2,6187181	0,0	0,000000e+00	0,06444	0,73361614
Pct400	0,0004698	1,456013e-03	0,04107	0,3548389	1,0	0,000000e+00	0,05127	0,39159424
Pct500	2,060e-06	6,924790e-05	0,0005159	0,2047163	0,0	0,000000e+00	0,01187	6,03434393
args	0,0	0,000000000	1,0	0,000000000	0,0	0,000000e+00	1,0	0,00000000

TAB. A.4 – Caractéristiques des 9 groupes résultants d'un partitionnement utilisant la méthode CLARA sur la troisième partie des données provenant du serveur de l'IRISA. Les variances sont calculées sur les variables centrées et réduites.

A.3 France Télécom

	Groupe 1		Groupe 2		Groupe 3		Groupe 4		Groupe 5	
Variables	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**
PctFérié	0,9297	1,626388e-01	0,4168	2,099685e-03	0,6390	1,07336648	0,04773	1,256290e-01	0,9187	9,119030e-02
Req-Jour	6,7198	1,652645e-04	14 250	4,863847e-01	177,6053	0,04060836	1,5971	7,710997e-06	6,8244	1,054184e-04
PctReq-Jour	1,396e-05	5,071708e-05	0,03750	1,631759e+01	4,658e-04	0,02024119	4,667e-06	6,357869e-06	1,592e-05	6,646455e-05
PctUserId	0,02536	1,355224e+00	0	0	0,1833	9,35501866	0	0	0,003817	1,168328e-01
PctGet	0,9984	1,025351e-02	0,9984	1,481886e-03	0,7869	5,08415033	0,9982	1,486603e-02	0,9845	3,871304e-01
PctPost	0	0	0	0	0,2003	19,2854838	0	0	0	0
PctHead	0,001612	1,575667e-02	0,00157	2,273734e-03	0,01280	0,15478134	0,001821	2,284477e-02	0,004254	5,559621e-02
PctVide	0	0	0	0	0	0	0	0	0	0
PctAutresMeth	0	0	5,401e-06	2,482711e-07	0	0	0	0	0,01122	3,427188e+00
PctHTTP/1.x	1	0	1	0	1	0	1	0	1	0
Pct200	0,9936	9,853810e-03	0,16100	3,187737e-01	0,6765	1,13018838	0	0	0,008472	1,487632e-02
Pct300	0,002619	3,135400e-03	0,8384863	4,399063e-01	0,053978	0,12193062	0	0	0,007761	2,278178e-02
Pct400	0,003764	4,706837e-03	0,0005108	2,884671e-05	0,2641	0,97206134	1	0	0,9838	3,357953e-02
Pct500	7,597e-06	7,102805e-05	0	0	0,005432	12,56894971	0	0	0	0
args	0	0	0	0	1	0	0	0	0	0

	Groupe 6		Groupe 7		Groupe 8		Groupe 9	
Variables	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**	**Moy.**	**Var.**
PctFérié	0,1817	3,772732e-01	0,4234	1,856563e-01	0,4437	0,5224011403	0,7481	2,985615e-01
Req-Jour	3,2718	2,563345e-05	13,4596	9,640507e-05	204,6304	0,0278080103	11,6226	9,723796e-05
PctReq-Jour	1,494e-05	6,807342e-05	2,491e-04	3,973984e-03	4,255e-04	0,0099160030	2,601e-05	4,319166e-05
PctUserId	0	0	0	0	0,01591	0,9463763698	0,01667	1,011765e+00
PctGet	0,9818	2,294088e-01	0,1850	1,473218e+00	0,9978	0,0443583794	1	0
PctPost	0	0	0	0	0	0	0	0
PctHead	0,01818	3,525347e-01	0,7715	3,832895e+00	0	0	0	0
PctVide	0	0	0	0	0	0	0	0
PctAutresMeth	0	0	0,04348	1,609102e+01	0,002166	0,4286799002	0	0
PctHTTP/1.x	1	0	1	0	1	0	1	0
Pct200	0,9911	1,368599e-02	0,2768	8,513788e-01	0,2543	0,0967553353	0,5981	6,302926e-02
Pct300	0,004572	8,823048e-03	0,01578	1,654887e-02	0,7446	0,1297043499	0,3915	8,982525e-02
Pct400	0,00271	4,420342e-03	0,7074	7,215748e-01	0,001037	0,0007800506	0,01042	2,259476e-02
Pct500	0,001574	3,639174e+00	0	0	0	0	0	0
args	0	0	0	0	0	0	0	0

Bibliographie

[1] T. Abbes, A. Bouhoula, and M. Rusinowitch. On the Fly Pattern Matching For Intrusion Detection with Snort. *Annals of telecommunications*, 59(9-10), sept.-oct. 2004.

[2] M. Almgren, H. Debar, and M. Dacier. A Lightweight Tool for Detecting Web Server Attacks. In *Proceedings of the 2000 ISOC Symposium on Network and Distributed Systems Security*, pages 157–170, 2000.

[3] Debra Anderson, Thane Frivold, and Alfonso Valdes. Next-Generation Intrusion Detection Expert System (NIDES) - A Summary. Technical Report SRI-CSL-95-07, SRI, Menlo Park, CA, May 1995.

[4] James P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, Fort Washington - Technical Report Contract 79F26400, 1980.

[5] Chris Anley. Advanced SQL Injection in SQL Server Applications. Technical report, Next Generation Security Software Ltd, 2002.

[6] A. Avizienis and L. Chen. On the Implementation of N-version Programming for software Fault Tolerance During Execution. In *Proceedings of the IEEE COMPSAC 77*, pages 149–155, Chicago, IL, USA, November 1977.

[7] Stefan Axelsson. Visualising intrusions : Watching the webserver. In *Proceedings of the 19th IFIP International Information Security Conference*, Toulouse, France, August 2004. Kluwer.

[8] Stefan Axelsson. *Understanding Intrusion Detection Through Visualisation*. PhD thesis, Chalmers University of Technology, Göteborg, Sweden, January 2005.

[9] E. Lundin Barse and E. Jonsson. Extracting Attack Manifestations to Determine Log Data Requirements for Intrusion Detection. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, Tucson, Arizona, USA, December 2004. IEEE Computer Society.

[10] Maxime Crochemore and Wojciech Rytter. *Text algorithms*. Oxford University Press, 1994.

[11] H. Debar, M. Becker, and D. Siboni. A Neural Network Component for an Intrusion Detection System. In *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 240–250, Piscataway, New Jersey, 1992.

[12] Hervé Debar. Habilitation à Diriger des Recherches – Détection d'Intrusions : Vers un Usage Réel des Alertes, Septembre 2004.

[13] Hervé Debar and Elvis Tombini. Accurate and fast detection of HTTP attack traces in web server logs. In *Proceedings of 14th EICAR Annual Conference*, Malta, May 2005.

[14] Hervé Debar and Elvis Tombini. WebAnalyzer : Détection précise d'attaques contre les serveurs HTTP. In *Proceedings of the 4th conference on Security and Network Architectures (SAR'05)*, Batz-sur-Mer, France, June 2005.

[15] Dorothy E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2) :222–232, 1987.

[16] Luca Deri. Improving Passive Packet Capture : Beyond Device Polling. In *Proceedings of the 4th International System Administration and Network Engineering Conference (SANE)*, Amsterdam, The Netherlands, October 2004.

[17] C. Dousson. *Suivi d'évolutions et reconnaissance de chroniques*. PhD thesis, 1994.

[18] R. Durst, T. Champion, B. Witten, E. Miller, , and L. Spagnuolo. Test and Evaluation of Computer Intrusion Detection Systems. In *Communications of the ACM*, volume 42, July 1999.

[19] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL : An Attack Language for State-based Intrusion Detection. In *Proceedings of the* 1^{st} *ACM Workshop on Intrusion Detection Systems*, Athens, Greece, November 2000.

[20] S.T. Eckmann, G. Vigna, and R.A. Kemmerer. STATL : An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2) :71–104, 2002.

[21] R. Fieldings, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.

[22] E. Fredkin. Trie memory. *Communication of the ACM*, 3(9) :490–499, September 1960.

[23] M.P. Georgeff and F. Ingrand. Real-Time Reasoning : The Monitoring and Control of Spacecraft. In *Proceedings of the 6th Conference on Artificial Intelligence Applications*, Santa Barbara, CA, 1990.

[24] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring – WEBDAV. RFC 2518, February 1999. Proposed standard.

[25] N. Habra, B. Le Charlier, I. Mathien, and M. Abdelaziz. ASAX : Software Architecture and Rule-Based Language for Univeral Audit Trail Analysis. In *Proceedings of the Second European Symposium on Research in Computer Security (Esorics 92)*, pages 435–450, November 1992.

[26] Mark Handley and Vern Paxson. Network Intrusion Detection : Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *USENIX Security Symposium*, Washington, DC, USA, August 2001.

[27] Paul Helman and Gunar Liepins. Statistical Foundations of Audit Trail Analysis fo the Detection of Computer Misuse. *IEEE Transactions on Software Engineering*, 19(9) :886–901, 1993.

[28] K. Ilgun, R.A. Kemmerer, and P.A. Porras. State Transition Analysis : A Rule-Based Intrusion Detection System. *IEEE Transactions on Software Engineering*, 21(3) :181–199, March 1995.

[29] R. Jagannathan, T. Lunt, D. Anderson, C. Dodd, F. Gilham, C. Jalali, H. Javitz, P. Neumann, A. Tamaru, and A. Valdes. System Design Document : Next-Generation Intrusion Detection Expert System (NIDES). Technical report, SRI, Menlo Park, CA, March 1993.

[30] Harold S. Javitz and Alfonso Valdes. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the Symposium on Research in Security and Privacy*, pages 316–326, Oakland, CA, May 1991.

[31] Harold S. Javitz, Alfonso Valdes, Teresa F. Lunt, Ann Tamaru, Mabry Tyson, and John Lowrance. Next Generation Intrusion Detection Expert System (NIDES) - 1. Statistical Algorithms Rationale - 2. Rationale for Proposed Resolver. Technical Report A016–Rationales, SRI International, 333 Ravenswood Avenue, Menlo Park, CA, March 1993.

[32] H.S. Javitz and A. Valdes. The NIDES Statistical Component Description and Justification. Technical report, SRI International, Menlo Park, CA, March 1994.

[33] M. St. Johns. Identification protocol. RFC 1413, February 1993. Proposed standard.

[34] A. Joshi, K. Joshi, and R. Krishnapuram. On Mining Web Access Logs. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 63–69, 2000.

[35] K. Julisch and M. Dacier. Mining Intrusion Detection Alarms for Actionable Knowledge. In *Proceedings of Knowledge Discovery in Data and Data Mining (SIGKDD)*, Edmonton, Alberta, Canada, July 2002.

[36] L. Kaufmann and P.J. Rousseeuw. Clustering Large Data Sets (with discussion). In E. S. Gelma and L. N. Kanal, editors, *Pattern Recognition in Practice II*, pages 425–437, North Holland/Elsevier, Amsterdam, 1986.

[37] L. Kaufmann and P.J. Rousseeuw. Clustering by Means of Medoids. In Y. Dodge, editor, *Statistical Data Analysis Based on the L_1 Norm*, pages 405–416, North Holland/Elsevier, Amsterdam, 1987.

[38] L. Kaufmann and P.J. Rousseeuw. *Finding Groups in Data – An Introduction to Cluster Analysis*. John Wiley, 2005.

[39] D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1972.

[40] C. Ko, M. Ruschitzka, and K. Levitt. Execution Monitoring of Security-Critical Programs in Distributed Systems : A Specification Based Approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 175–187, May 1997.

[41] C. Kruegel, D. Mutz, F. Valeur, and G. Vigna. On the Detection of Anomalous System Call Arguments. In *Proceedings of the 8th European Symposium on Research in Computer Security (ESORICS)*, LNCS, Norway, October 2003. SV.

[42] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *Proceedings of the* 10^{th} *ACM Conference on Computer and Communication Security (CCS '03)*, pages 251–261, Washington, DC, October 2003. ACM Press.

[43] C. Kruegel, G. Vigna, and W. Robertson. A Multi-model Approach to the Detection of Web-based Attacks. *Computer Networks*, 48(5) :717–738, August 2005.

[44] Josue Kuri, Gonzalo Navarro, and Ludovic Mé. Fast Multipattern Search Algorithms for Intrusion Detection. *Fundamenta Informaticae*, 56(1,2) :23–49, 2003.

[45] L. Lebart, A. Morineau, and M. Piron. *Statistique exploratoire multidimentionnelle*. Dunod, 2005.

[46] Sin Yeung Lee, Wai Lup Low, and Pei Yuen Wong. Learning fingerprints for a database intrusion detection system. In *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS)*, LNCS, Zurich, Switzerland, October 2002. SV.

[47] W. Lee, J. B. D. Cabrera, A. Thomas, N. Balwalli, S .Saluja, and Y. Zhang. Performance Adaptation in Real-Time Intrusion Detection Systems. In A. Wespi, G. Vigna, and L. Deri, editors, *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, number 2516 in Lecture Notes in Computer Science, Zurich, Switzerland, October 2002. Springer-Verlag.

[48] W. Lee and S.J. Stolfo. A Data Mining Framework for Building Intrusion Detection Model. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999.

[49] Wenke Lee and Dong Xiang. Information-Theoretic Measures for Anomaly Detection. In *Proceedings of The 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2001.

[50] U. Lindqvist and A. Porras. Detecting Computer and Network Misuse with the Production-Based Expert System Toolset (P-BEST). In *IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, CA, May 1999.

[51] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. Evaluating Intrusion Detection Systems : The 1998 DARPA Off-line Intrusion Detection Evaluation. In IEEE Computer Society Press, editor, *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX'00)*, volume 2, pages 12–26, Los Alamitos, CA., 2000.

[52] Wai Lup Low, Joseph Lee, and Peter Teoh. DIDAFIT : Detecting Intrusions in Databases Through Fingerprinting Transactions. In *Proceedings of the 4th International Conference on Enterprise Information Systems*, Ciudad Real, Spain, April 2002. Kluwer.

[53] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, P.G. Neumann, H.S. Javitz, A. Valdes, and T.D. Garvey. Real Time Intrusion Detection Expert System (IDES) - Final Report. Technical report, SRI International, Menlo Park, CA, February 1992.

[54] Teresa F. Lunt. Automated Audit Trail Analysis and Intrusion Detection : A Survey. In *Proceedings of the 11th National Computer Security Conference*, Baltimore, MD, USA, 1988.

[55] A. Luotonen. The Common Logfile Format. http ://www.w3.org/pub/WWW/Daemon/User/Config/Logging.html, 1995.

[56] F. Majorczyk, E. Totel, and L. Mé. COTS Diversity Based Intrusion Detection and Application to Web Servers. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, Lecture Notes in Computer Science, Seattle, Washington, USA, September 2005. Springer-Verlag.

[57] F. Majorczyk, E. Totel, and L. Mé. Détection d'intrusions par diversification de COTS. In *Proceedings of the 4th conference on Security and Network Architectures (SAR'05)*, Batz-sur-Mer, France, June 2005.

[58] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz. A Data Mining Analysis of RTID Alarms. In *Proceedings of the First International Workshop on the Recent Advances in Intrusion Detection (RAID99)*, Louvain-la-Neuve, Belgium, September 1999.

[59] J. McQueen. Some Methods for Classification and Analysis of Multivariate Observation. In L. Le Cam and J. Neyman, editors, *5th Berkeley Symp. Math. Star. Prob*, volume 1, pages 281–297, 1967.

[60] Jambu Michel. *Méthodes de base de l'analyse de données.* Eyrolles et France Télécom-CNET, 1999.

[61] Benjamin Morin and Hervé Debar. Correlation of Intrusion Symptoms : An Application of Chronicles. In G. Vigna, E. Jonsson, and C. Kruegel, editors, *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Lecture Notes in Computer Science, Pittsburgh, PA, USA, September 2003. Springer-Verlag.

[62] D.R. Morrison. PATRICIA - Practical Algorithm To Retrieve Information Coded In Alphanumeric. *JACM*, 15(4) :514–534, 1968.

[63] IETF Network Working Group. Hypertext Transfer Protocol – HTTP/1.0, rfc 1945, May 1996.

[64] N. Nuansri, S. Singh, and T. S. Dillon. A Process State-Transition Analysis and its Application to Intrusion Detection. In *Proceedings of the 15th Annual Computer Security Application Conference*, 1999.

[65] Vern Paxson. Bro : a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands : 1999)*, 31(23–24) :2435–2463, 1999.

[66] P. A. Porras. *STAT : A State Transition Analysis Tool for Intrusion Detection.* PhD thesis, Computer Science Departement, University of California, Santa Barbara, CA, June 1992.

[67] Jean-Philippe Pouzol and Mireille Ducassé. Formal Specification of Intrusion Signatures and Detection Rules. In *CSFW*, pages 64–. IEEE Computer Society, 2002.

[68] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service : Eluding network intrusion detection. Technical Report T2R-0Y6, Secure Networks, Inc., Calgari, Alberta, Canada, January 1998.

[69] Isidore Rigoutsos and Aris Floratos. Combinatorial pattern discovery in biological sequences : The teiresias algorithm [published erratum appears in bioinformatics 1998 ;14(2) : 229]. *Bioinformatics*, 14(1) :55–67, 1998.

[70] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of LISA'99*, pages 229–238, Seattle, Washington, USA, Nov 1999. USENIX Association.

[71] Shai Rubin, Somesh Jha, and Barton Miller. Automatic Generation and Analysis of NIDS Attacks. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, Tucson, Arizona, USA, December 2004. IEEE Computer Society.

[72] SunSoft. *SunSHIELD Basic Security Module Guide*. SunSoft, Mountain View, CA, 1995.

[73] H.S. Tenk, K. Chen, and S. Lu. Security Audit Trail Analysis Using Inductively Generated Predictive Rules. In *Proceedings of the 6th Conference on Artificial Intelligence Applications*, pages 24–29, Piscataway, New Jersey, Mars 1990.

[74] E. Tombini, H. Debar, and B. Morin. *Procédé de caractérisation automatique de paramètres à partir de leurs valeurs associés appliqué à la détection d'intrusion*. France Télécom R&D, 2005.

[75] Elvis Tombini, Hervé Debar, Ludovic Mé, and Mireille Ducassé. A Serial Combination of Anomaly and Misuse IDSes Applied to HTTP Traffic. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, Tucson, Arizona, USA, December 2004. IEEE Computer Society.

[76] Security Tracker. Vulnerability Statistics April 2001 - March 2002. http ://www.securitytracker.com/learn/statistics.html, April 2002.

[77] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection. In *Proceedings of the IEEE Infocom Conference*, pages 333–340, March 2004.

[78] A. Valdes and K. Skinner. Probabilistic Alert Correlation. In Wenke Lee, Ludovic Mé, and Andreas Wespi, editors, *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, number 2212 in Lecture Notes in Computer Science, Davis, CA, USA, October 2001. Springer-Verlag.

[79] F. Valeur, D. Mutz, and G. Vigna. A Learning-Based Approach to the Detection of SQL Attacks. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, Vienna, Austria, July 2005.

[80] G. Vigna, S.T. Eckmann, and R.A. Kemmerer. The STAT Tool Suite. In *Proceedings of DISCEX 2000*, pages 46–55, Hilton Head, SC, January 2000. IEEE Press.

[81] G. Vigna, S. Gwalani, K. Srinivasan, E. Belding-Royer, and R. Kemmerer. An Intrusion Detection Tool for AODV-based Ad Hoc Wireless Networks. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 16–27, Tucson, AZ, December 2004.

[82] G. Vigna, W. Robertson, V. Kher, and R.A. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, pages 34–43, Las Vegas, NV, December 2003.

[83] J. Viinikka and H. Debar. Monitoring IDS Background Noise Using EWMA Control Charts and Alert Information. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, Lecture Notes in Computer Science, Nice, France, September 2004. Springer-Verlag.

[84] D. Wagner and D. Dean. Intrusion Detection via Static Analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001. IEEE Press.

[85] Rex Walters. Tracking Hardware Configurations in a Heterogeneous Network with syslogd. In *LISA '95 : Proceedings of the 9th USENIX conference on System administration*, pages 241–246, Berkeley, CA, USA, 1995. USENIX Association.

[86] Christina Warrender, Stephanie Forrest, and Barak A. Pearlmutter. Detecting Intrusions using System Calls : Alternative Data Models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[87] Andreas Wespi, Marc Dacier, and Hervé Debar. An intrusion detection system based on the Teiresias pattern-discovery algorithm. In Urs E. Gatiker, Pia Pedersen, and Karsten Petersen, editors, *Proceedings of EICAR'99*, Aalbord, Denmark, February 1999. European Institute for Computer Anti-Virus Research.

[88] Mark Wood and Mike Erlinger. Intrusion Detection Message Exchange (IDMEF) requirements. Internet Engineering Task Force - IDWG, 2003. Work in progress, expires April 22nd, 2003.

[89] Ke Zhang, Amy Yen, Xiaoliang Zhao, Daniel Massey, Shyhtsun Felix Wu, and Lixia Zhang. On Detection of Anomalous Routing Dynamics in BGP. In Nikolas Mitrou, Kimon P. Kontovasilis, George N. Rouskas, Ilias Iliadis, and Lazaros F. Merakos, editors, *NETWORKING*, volume 3042 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 2004.

Résumé

Cette thèse s'intéresse à l'amélioration des techniques de détection d'intrusions. Cependant, nous considérons que l'amélioration d'une technique particulière se heurte rapidement à la recherche d'un compromis optimal entre le problème des faux positif et des faux négatif. Ceci ne permet pas d'améliorer simultanément les deux mesures.

Dans cette thèse nous présentons un outil de détection d'intrusions spécifique aux serveurs web, basé sur la combinaison en série d'un outil de détection d'intrusions comportementale et d'un outil de détection d'intrusions par scénarios. La combinaison de méthodes de détection d'intrusions permet de tirer partie des capacités de qualification de chacune des méthode utilisées.

Nous montrons que parmi les combinaisons possibles de méthodes de détection d'intrusions comportementale et par scénarios, la plus adaptée à nos besoins et à notre contexte est la combinaison $A_u \rightarrow M$. Cette combinaison en série d'une méthode de détection d'intrusions comportementale, suivie d'une méthode de détection d'intrusions par scénarios appliquée aux événements non reconnus comme sains, nous permet de qualifier l'ensemble des événements et de découvrir de nouvelles attaques.

Nous avons mis en œuvre cette combinaison en construisant un outil de détection d'intrusions comportementale, *WebFilter*, et un outil de détection d'intrusions par scénarios, *WebAnalyzer*.

La construction du modèle comportemental de *WebFilter* se base sur des méthodes statistiques connues et éprouvées. Elle nous a permis d'établir des profils de comportement utilisable pour d'autres serveurs web cibles. Les modèles de comportement ainsi créés ne génèrent que peu ou pas de faux négatifs et qualifie de 54% à 94% des requêtes vers les serveurs web que nous avons analysés.

L'analyse effectuée par la méthode de détection d'intrusions par scénarios utilise plus de 650 signatures. L'efficacité de cette méthode réside d'une part dans l'utilisation de la combinaisons des signatures activées pour établir un diagnostic complet. D'autre part, l'attribution d'une sévérité à une alerte permet de hiérarchiser les alertes présentées à l'opérateur. Ces deux caractéristiques facilitent l'interprétation de la nocivité d'une attaque par un opérateur de sécurité et la mise en place de contre-mesures efficaces.