Détection d'intrusions : de l'utilisation de signatures statistiques

Payas Gupta *, Chedy Raïssi **, Gérard Dray **, Pascal Poncelet **, Johan Brissaud ***

*LNMIIT - Jaipur (Raj.) 302015 - India
payasgupta@gmail.com
**LGI2P - École des Mines d'Alès, Parc Scientifique G. Besse, 30035 Nîmes, France
Chedy.Raissi@ema.fr, Gerard.Dray@ema.fr, Pascal.Poncelet@ema.fr
***BeeWare SA
Le Millenium, Bâtiment B, 501 rue Denis Papin 34000 Montpellier
jbrissaud@bee-ware.net

Résumé. Garantir la sécurité des serveurs web devient un enjeu majeur pour les entreprises ou les organisations et il devient de plus en plus difficile de détecter parmi les différentes requêtes celles qui correspondent à un comportement normal de celles qui correspondent à un comportement malveillant. Même s'il existe des Systèmes de Détection d'Intrusions (SDI), ces derniers ne sont malheureusement pas ou plus adaptées aux nouvelles attaques. Motivés par ce constat, les chercheurs de la communauté fouille de données s'intéressent de plus en plus à la détection de fraudes dans les réseaux. Dans cet article nous proposons une nouvelle approche de détection de fraudes basée sur un paramétrage automatique du comportement des requêtes normales et de la distribution de valeurs d'attributs. Les connaissances extraites représentées sous la forme de signatures statistiques peuvent alors être utilisées pour rechercher efficacement des comportements malveillants dans le flot de requêtes. Enfin, de manière à prendre en compte les nouveaux services et minimiser les fausses alarmes, nous maintenons de manière incrémentale les signatures. L'approche proposée a été expérimentée sur des jeux de données réelles et a montré une précision de plus de 99.98% pour la prédiction de requêtes valides et un taux de faux positifs de moins de 9.88%.
Keywords: Détection d'intrusions, signatures statistiques, approche incrémentale.

1 Introduction

Le déploiement des ordinateurs et des réseaux a considérablement augmenté les risques causés par les attaques sur les systèmes informatiques qui deviennent un réel problème pour les entreprises et les organisations. Par exemple, une étude récente du National Institute of Standards and Technology a montré que les dommages, pour les compagnies américaines, étaient estimés à plus de 59,6 millions de dollars par an. Alors qu'auparavant de nombreuses attaques se focalisaient sur les serveurs Web car ils étaient souvent mal configurés ou mal maintenus, les attaques les plus récentes profitent des failles de sécurité des services ou applications Web qui sont plus vulnérables. Pour pallier ce problème, de nouvelles approches appelées Systèmes de Détection d'Intrusions (SDI) ont fait leur apparition. Ces outils, installés sur les réseaux, ont pour objectif d'analyser le trafic de requêtes et de détecter des comportements malveillants. Ils peuvent être classés en deux grandes catégories (e.g. McHugh et al. (2000); Proctor (2001)) : les *systèmes de détection d'anomalies* et les *systèmes de détection d'abus*.

Le principe des approches de détection d'abus consiste à appliquer des techniques d'apprentissage sur des attaques connues de manière à en définir leurs signatures. Ensuite, à l'aide d'expressions régulières

ou de correspondance de motifs ces dernières sont utilisées pour reconnaître les attaques dans les flots de requêtes. Si ces approches sont efficaces pour reconnaître les attaques passées, elles sont malheureusement mises en défaut lorsque de nouvelles attaques interviennent et certains logiciels profitent de cette faille pour passer outre ces systèmes de détection Fogla et Lee (2006). D'un autre côté, les systèmes de détection d'anomalies s'intéressent à l'analyse des comportements normaux, i.e. les comportements valides sur le site, et cherchent à les caractériser. Ils considèrent ainsi qu'une intrusion correspond à un comportement qui dévie de la norme. Traditionnellement, les motifs sont obtenus à l'aide de techniques d'apprentissages supervisés ou non. Par exemple, dans Giacinto et al. (2006), les auteurs utilisent un ensemble d'apprentissage non étiqueté et classent les requêtes en attaque ou non. Les travaux de Cohen et al. (2004) considèrent un ensemble de données étiquetées "attaque" ou "non attaque" et appliquent des algorithmes de clustering pour déterminer les différentes classes.

Par rapport aux détection d'abus, ces systèmes ont l'avantage d'être moins dépendants des attaques extérieures. Par contre, ils ne sont pas capables de prendre en compte les comportements non prévus ou les évolutions des applications du serveur Web et peuvent ainsi engendrer un grand nombre de fausses alarmes. Pour résoudre ce problème, de nouvelles approches ont été proposées et tentent de maintenir les signatures de manière automatique (e.g. Esposito et al.; Li (2005); Yeung et Ding (2003)). Des techniques basées sur la logique floue, les algorithmes génétiques ou les réseaux de neurones sont fréquemment utilisées (e.g. de Sá Silva et al. (2007); Saniee et al. (2007); Newsome et al. (2005)) mais, à cause de la structure complexe des requêtes, elles sont généralement difficiles à mettre en œuvre et sont souvent pénalisées par des temps de réponses prohibitifs. Aussi, l'évolution des signatures se résume très souvent à l'intervention d'un expert du domaine et les changements sont lents et coûteux Adeva et Atxa (2007); Paxson (1999).

Il est donc nécessaire de proposer des systèmes qui soient capables d'apprendre automatiquement les signatures des requêtes valides mais qui soient aussi capables de les maintenir afin de réduire le nombre de fausses alarmes Pietraszek et Tanner (2005). Dans cet article, nous proposons un SDI basé sur la méthode de détection d'anomalies. Dans ce contexte, nous souhaitons caractériser et modéliser les comportements typiques de serveurs Web Pereira et al. (2007). Notre objectif est de nous focaliser plus particulièrement sur la détection d'attaques nouvelles et sur les modifications apportées aux anciennes attaques qui ne peuvent pas être détectées par les SDI actuels. Aussi, nous considérons dans la suite de cet article que les anciennes attaques peuvent être traitées par un SDI traditionnel. Etant donné que chaque serveur Web possède ses propres caractéristiques ou ses propres usages, nous déterminons, pour un serveur, les caractéristiques qui correspondent à des requêtes valides, i.e. nous modélisons les données valides sous la forme d'attributs qui pourront être utilisés pour générer des signatures. Ces signatures seront alors utilisées pour détecter les comportements anormaux sur le serveur Web. Bien entendu, cette approche peut être utilisée pour détecter les attaques de n'importe quel serveur mais le jeu d'apprentissage d'un serveur ne peut pas être utilisé pour tester les requêtes d'un autre serveur.

Le reste de l'article est organisé de la manière suivante. Dans la section 2 nous discutons les travaux antérieurs. Une présentation générale de notre approche est proposée dans la section 3 et poursuivie par une description des algorithmes dans la section 4. La section 5 décrit quelques unes des expériences réalisées sur des jeux de données réels. La conclusion est proposée dans la section 6.

2 Travaux antérieurs

Notre proposition est proche des méthodes de détections d'anomalies. Le premier SDI basé sur les anomalies a été introduit par D.E. Denning (1987) et de nombreux travaux ont été réalisés dans ce domaine. Les différences essentielles sont liées à la manière de modéliser les données, i.e. de caractériser les requêtes valides. Proche de nos préoccupations, des travaux récents se sont intéressés à la détection d'intrusions dans des applications Web. Dans Wagner et Dean (2001), les auteurs montrent comment une

analyse statique des applications Web peut être utilisée pour dériver de manière automatique un modèle du comportement des applications. Cette technique améliore l'approche de Denning (1987) en offrant un haut degré d'automatisation, une protection contre une grande classe d'attaques basées sur des erreurs de programmation, et l'élimination de fausses alarmes. Elle est cependant très dépendante des langages de programmation utilisés pour développer les applications Web. Dans Robertson et al. (2006), les auteurs précisent les limitations des systèmes de détections d'anomalies utilisant des techniques de caractérisation et de généralisation. Ils proposent dans Kruegel et al. (2005), de représenter le comportement des requêtes normales par des attributs. Ces derniers sont générés de manière à distinguer les requêtes valides de requêtes invalides et nécessitent que l'ensemble d'apprentissage soit uniquement composé de requêtes valides. Même si notre proposition utilise également des attributs pour décrire des requêtes normales, elle ne nécessite pas de spécifier des seuils de valeurs qui sont souvent difficiles à déterminer. En outre, à partir de ces attributs nous générons un ensemble de signature réduit à partir duquel nous pouvons analyser les nouvelles requêtes du serveur.

D'autres travaux (e.g. Lee et al. (2001)) ont proposé des solutions pour générer en temps réel les signatures mais malheureusement les expérimentations montrent qu'elles produisent un trop grand nombre de fausses alarmes. Pour limiter les effets des fausses alarmes produites par les méthodes d'anomalies de détection, nous introduisons une nouvelle approche qui permet non seulement de générer des signatures de comportement de requêtes valides mais qui en plus permet de maintenir de manière incrémentale ces signatures quand cela est nécessaire. Bien que le modèle présenté dans cet article soit spécifique à des logs de serveur Web, il peut être adapté facilement à d'autres types de comportements inattendus, e.g. logs de serveurs Proxy. La seule différence réside dans le choix des attributs qui caractérisent des requêtes valides.

3 Le système de détection d'anomalies

Chaque serveur possède ses propres caractéristiques, i.e. les requêtes parvenant à un serveur sont usuellement similaires. Bien entendu, cela peut varier d'un serveur à un autre. En utilisant les caractéristiques d'un serveur Web, cette approche a pour objectif de modéliser les requêtes par différents attributs qui permettent ensuite de générer des signatures statistiques.

3.1 Description des données

Notre approche de détection d'anomalies analyse les requêtes HTTP telles quelles sont stockées par la plupart des serveurs Web (e.g., Apache (2007)). Ces requêtes respectent bien évidemment le format des caractères défini par le RFC 1738 Berners-Lee et al. (1994) : un encodage des caractères est possible à l'aide du caractère % suivi du code ASCII du caractère à coder en notation hexadecimal. Par souci de simplification, dans cet article, nous nous intéressons aux requêtes qui utilisent des arguments pour passer des valeurs aux programmes situés du côté du serveur. Plus formellement, nous considérons par la suite comme entrée du processus un ensemble ordonné d'Uri $U = \{u_1, u_2, ...u_n\}$ où chaque Uri u_i peut être exprimé sous la forme du chemin à la ressource demandée (*path*) et d'une chaîne requête optionnelle (*query*). Cette chaîne est utilisée pour transmettre des valeurs d'arguments à la ressource demandée et elle est identifiée par le caractère ' ?'. Plus formellement, une query q est une liste ordonnée de tuple $< (a_1, v_1), (a_2, v_2), ... (a_n, v_n) >$ où a_i correspond à un nom d'attribut et v_i correspond à la valeur de cet attribut.

L'exemple suivant illustre un exemple d'entrée dans un log simplifié de serveur Web.

192.233.57.105 - jean [15/Sep/2007 :23 :59 :59 - 0800] "GET /scripts/access.cgi ?user=jean&cred=admin" 200 2123

La partie $/scripts/access.cgi$ correspond au path et la partie query est composée de $< (user, jean), (cred, admin) >$.

Nous utiliserons, par la suite, les termes Uri pour désigner la partie *path* (e.g. $/scripts/access.cgi$) et *Query* pour la partie valeur des arguments de la requête (e.g. $jean, admin$).

3.2 Extraction des Uri et des arguments de la requête

Le principe général de notre système de détection est de détecter à partir d'un ensemble de données valides, les attributs qui peuvent les caractériser et ce d'une manière statistique rapide. Pour chaque requête du fichier log valide, nous séparons la partie Uri de la partie Query.

En outre chaque *Query* est elle même décomposée en autant de partie qu'il existe d'arguments dans la requête. Etant donné qu'une attaque peut intervenir sur différentes parties d'une requête, notre objectif dans ce prétraitement est de séparer chaque élément de manière à ce qu'il ne puisse pas influencer les autres. Ils n'existent donc pas de liaisons entre les éléments et les traitements peuvent donc être réalisés séparément.

3.3 Classification des attributs

En fait, le choix des attributs est important dans la mesure où ils vont permettre de différencier des requêtes valides de requêtes invalides. Ils doivent donc être choisis pour produire le moins possible de faux positifs ou de faux négatifs. Certains attributs sont calculés sur la chaîne originale et d'autres sur la chaîne décodée. Une chaîne est décodée récursivement jusqu'à ce qu'elle ne puisse plus l'être (C.f. Tableau 1). Ce traitement permet d'aider à identifier des attaques qui utilisent un encodage, double ou de niveau supérieur, pour envoyer des attaques Mangarae et Morganti (2007).

Bien entendu, ces attributs sont spécifiques d'un serveur Web et différents tests sont nécessaires pour en retenir l'ensemble pertinent. Nous présentons ci-dessous quelques uns des attributs que nous avons retenus pour les expérimentations finales (la sélection ayant été faite avec l'aide d'un expert dans le domaine de la securité informatique) en explicitant les raisons de nos choix.

1. *Nombre de caractères %00 (NULL)* : calculé sur la chaîne originale extraite des parties Uri et Query, cet attribut est utile pour détecter les attaques qui utilise le caractère NULL pour ignorer les caractères suivants. Par exemple, dans la chaîne $location = /etc/passwd\%00dfgdf$ le navigateur ignorera *dfgdf*. Lors des expérimentations, cet attribut absent des requêtes valides apparaissait dans plus de 20% des attaques.
2. *Nombre de caractères codés* : indique, dans la chaîne, le nombre de caractères codés.
3. *Nombre de caractères codés qui ne nécessitent pas d'encodage* : indique les caractères qui sont encodés soit par l'utilisateur soit par le navigateur pour la transmission.
4. *Nombre de caractères codés au moins deux fois*

	Chaîne
Chaîne originale	SELECT%2B%252A% 2BFROM%2B%2560admin%2560
Après le 1^{er} décodage	SELECT+%2A+FRO M+%60admin%60
Après le 2^{nd} décodage	SELECT * FROM 'admin'

TAB. 1 – *Un exemple d'encodage*

Les attributs 2-4 sont calculés sur la chaîne originale, i.e. sans décodage. Ils sont très informatifs car ils caractérisent les principes d'encodage utilisés sur un site. Dans un comportement normal le nombre de caractères encodé est traditionnellement faible dans une requête à part pour des caractères régionaux particuliers : *è*, *â*, *é*, *õ* etc.).

	Chaîne
Chaîne Originale	%2573%2565%256C%2 565%2563%2574%2B% 252A%2B%2566%2572 %256F%256D%2B%25 60%2561%256D%2569 %256E%2560
Après le 1^{er} décodage	%73%65%6C%65%63% 74+%2A+%66%72%6F %6D+%60%61%6D%6 9%6E%60
Après le 2^{nd} décodage	select * from 'admin'

TAB. 2 – *Un exemple d'encodage*

Les Tableaux 1 et 2 illustrent les raisons pour lesquelles un grand nombre d'attaques encodent les données. En effet, via cet encodage, les attaques peuvent passer au travers de nombreux SDI qui se limitent au premier ou au second niveau.

Les autres paramètres retenus correspondent principalement aux nombres de caractères spéciaux, d'espace, de lettre de l'alphabet, de chiffres, ... Ces paramètres sont calculés sur la chaîne décodée.

3.4 Génération des valeurs des attributs

Après l'étape de classification, les valeurs des attributs sont calculées pour chacune des entrées obtenues dans la section 3.2. Ainsi, par exemple, pour la requête suivante :

$/\%63alendrier\%27\%3B\%2A/../../?var = 3649439\%5C4\%7C\%20\%258A8\&res = qsddff$

et après décomposition des différentes parties, nous obtenons :

Uri	$/\%63alendrier\%27\%3B\%2A/../../$	0 4 1 0 1 21 1 1 1 4 0 0 0 2 ... 0 0 0
Query-val$_1$	$3649439\%5C4\%7C\%20\%258A8$	0 4 0 1 2 13 0 0 0 0 1 0 0 0 ... 1 1 0
Query-val$_2$	$qsddff$	0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 ... 0 0 0

Dans cet exemple, la valeur 4 dans "0 4 0 1 2 13 0 0 0 0 1 0 0 0 ... 1 1 0" correspond à l'attribut "Nombre de caractères codés".

3.5 Génération de la distribution

Après avoir calculé les valeurs de tous les attributs pour chacune des parties de la requêtes, nous calculons pour chacun des attributs la moyenne (μ) et l'écart type (σ) et ceci de manière séparée pour l'Uri et les valeurs d'arguments de la requête[1]. Ainsi, pour chaque attribut, nous pouvons déterminer son intervalle de validité : de μ-σ à μ+σ comme illustré dans le tableau 3.

3.6 Création des signatures

Après avoir générés les différentes valeurs de distribution pour l'ensemble des données valides, le même jeu de données est réutilisé et pour chaque requête nous examinons si la valeur de chaque attribut de la requête appartient à son intervalle de validité. Si elle appartient à l'intervalle choisi, elle est codée par *1* autrement par *0*. Ce codage correspond à la signature de la requête valide et cette procédure est appliquée à toutes les requêtes du jeux de données (C.f. tableau 4).

Nous ne retenons au final que les signatures uniques qui seront codées sous la forme de vecteurs de bits pour des raisons d'efficacité. A l'aide de ces signatures, les nouvelles requêtes sur le serveur Web peuvent être testées pour rechercher des comportements malveillants, i.e. qui n'appartiennent pas à l'ensemble des signatures.

[1]Dans cet article, par souci de simplification, nous considérons que les distributions pour les requêtes non-malveillantes sont normales.

Partie Uri		**Partie Query Argument**	
μ-σ	μ+σ	μ-σ	μ+σ
0.1540	0.1658	-0.5233	0.5371
-0.0032	0.0032	-0.1316	0.1334
-0.0007	0.0007	-0.2104	0.2136
-0.0413	0.0451	-0.0278	0.029
22.4660	49.2036	-1.6920	19.1362

TAB. 3 – *Un exemple d'intervalle de validité pour les 5 premiers attributs.*

00110110101111...1100
10110101111110...1110
10110101111111...1100
10110101111110...0000
10110101111111...0100
10110101111111...1000
11111111111111...1111

TAB. 4 – *Un exemple de signatures valides*

3.7 Analyse des signatures

La génération des signatures est très importantes car elles caractérisent les requêtes normales dans le modèle. Elles montrent que certains paramètres des requêtes n'appartiennent pas à l'intervalle de distribution et sont donc codées avec *0*. Ceci n'implique cependant pas que la requête soit invalide. Il n'est pas nécessaire pour les valeurs de tous les paramètres d'une requête valide qu'elles apparaissent complètement dans leur intervalle. Supposons que nous n'ayons pas de signatures et que nous calculions simplement l'intervalle. Ainsi, lorsqu'une nouvelle requête arrive nous calculons ses valeurs d'attributs et vérifions si elles appartiennent ou non à la distribution. Si elles n'appartiennent pas à l'intervalle de distribution, nous augmentons son poids. Cependant comme nous l'avons précisé précédemment, il n'est pas nécessaire pour une requête valide de toujours satisfaire tous les intervalles des paramètres. Dans ce cas, cela produirait de nombreuses fausses alarmes car nous ne donnons pas d'importance à ces valeurs qui sont hors de l'intervalle. Ces signatures sont appelées Signatures Statistiques car, basées sur la distributions de la valeur des attributs, elles sont générées et représentent les caractéristiques du serveur Web.

4 Algorithmes

Le processus général est divisé en deux phases principales : la *phase de paramétrage* et la *phase de test*. Au cours de la première phase, le SDI est exécuté avec des requêtes valides et les signatures correspondantes au serveur Web sont générées. La phase de test du SDI examine toutes les nouvelles requêtes venant du serveur Web et sépare les requêtes valides des invalides. Une alarme est déclenchée quand le système détecte une requête supposée être une attaque ou une requête invalide par rapport aux comportements modélisés. La troisième étape de notre approche correspond à la maintenance des signatures.

4.1 Phase de paramétrage

La phase de paramétrage est une étape importante pour chaque SDI car les résultats obtenus dans la phase de test dépendent de la manière dont le modèle du SDI a été construit et du type de données utilisées. Comme nous l'avons vu dans les sections précédentes, nous réduisons les requêtes aux parties Uri et Query aussi l'algorithme ne considère que ces deux composantes mais peut aisément être étendu à tous les champs de la partie entête de la requête. Considérons l'ensemble L composé de requêtes valides R. μ_i et σ_i sont les moyennes et écarts types pour le i^{eme} attribut de toutes les requêtes de la phase de paramétrage. Afin de s'adapter à un principe de paramétrage *on-line*, les valeurs μ_i et σ_i sont obtenus de manière incrémentale à l'aide des équations 1 et 2.

$$\mu_1 = x_1, \qquad \mu_{k+1} = \frac{k}{k+1}\mu_k + \frac{1}{k+1}x_{k+1} \tag{1}$$

$$\sigma_1 = 0, \qquad \sigma_{k+1} = \sqrt{\frac{k}{k+1}\sigma_k^2 + \frac{k}{(k+1)^2}(\mu_k - x_{k+1})^2} \tag{2}$$

Dans la Fonction 1, à chaque fois qu'une nouvelle requête est traitée, les valeurs correspondantes aux attributs sont calculées et les μ et σ associés sont mis à jour. Ceci est généralisé à toutes les requêtes R de l'ensemble des données valides L, *compute_signature(type)* est utilisé pour générer, de manière séparée pour les Uri et les Query, les signatures.

Fonction `1 Parametering_phase(`L`)`

Data : L=Ensemble valide de requêtes R.
Result : Signatures pour Uri et Query.
begin
  for $i \leftarrow 1$ **to** $no\_of\_attributes$ **do**
    $\mu_i \longleftarrow 0$, $\sigma_i \longleftarrow 0$
  while $\exists R \in L$ **do**
    if $Uri \in R$ **then**
      U $\longleftarrow$ extract Uri
      compute attribute values
      compute_mu_sigma(U)
    if $Query \in R$ **then**
      Q $\longleftarrow$ extract value from the query arg
      while $\exists Q$ **do**
        compute attribute values
        compute_mu_sigma(Q)
  compute_signature(U)
  compute_signature(Q)
end

4.2 Phase de test

Dans cette phase, la Fonction 2 extrait de la nouvelle requête, les composantes Uri et Query et la valeur de tous les attributs est calculée. Si la valeur d'un attribut appartient à son intervalle calculé dans la phase de paramétrage, i.e. (μ-$\sigma \leq$ val $\leq \mu$+σ), il est codé *1* autrement *0*. Ce processus est répété pour tous les attributs et cela forme la signature de la partie Uri et Query.

A la fois pour les parties Uri et Query, la nouvelle signature est comparée avec l'ensemble des autres signatures obtenues lors de la phase de paramétrage. Pour chaque signature, le nombre de bits différent est compté et la somme est divisée par le nombre d'attributs. Ce nombre correspond au poids ($0 \leq$ weight ≤ 1) donné à la requête pour une signature. Ceci est réalisé pour toutes les signatures issues de la phase de paramétrage. Finalement, le minimum de tous les poids est calculé. Dans le cas de l'Uri ce nombre minimum montre de combien la nouvelle signature de l'Uri est différente des signatures statistiques. Par contre, dans le cas de la composante Query, cette procédure est répétée pour chaque argument et le maximum de tous les poids minimum est calculé à chaque fois pour décrire la différence par rapport aux signatures statistiques.

Poids de la partie Uri	**Poids de la partie Query**
0.166667	
0.166667	
0.000000	
0.000000	0.000000
0.000000	0.055556

TAB. 5 – *Un exemple de tableau de valeurs de poids*

De manière à illustrer pourquoi nous retenons le maximum de tous les poids minimum dans la composante Query, considérons le cas suivant. La composante Query contient trois arguments, les deux premiers sont valides et le dernier correspond à une attaque. Pour chacun des arguments, nous recherchons la correspondance la plus proche par rapport aux signatures statistiques et nous calculons le poids associé. Pour les deux premiers arguments, ce poids sera de 0 (correspondance parfaite) et pour le troisième, le poids aura une valeur supérieure à 0. Ainsi, quand nous considérons le maximum de tous les poids nous savons exactement dans quelle partie de la requête l'attaque a eu lieu. Le tableau 5 illustre les poids obtenus pour cinq requêtes. Si la requête ne contient pas de paramètres alors la valeur est évidemment nulle, e.g. les trois premières requêtes dans le tableau. Pour chaque requête, s'il existe un poids supérieur à zéro alors cela indique que la requête complète ne correspond pas aux signatures qui ont été générées et il peut s'agir d'une attaque.

4.3 Mise à jour des signatures

Comme nous pouvons le constater à la fin de la Fonction 2, l'approche peut être implémentée de manière incrémentale et les signatures peuvent être mises à jour de manière très efficace. Pour maintenir les connaissances de manière incrémentale, les valeurs des attributs et les μ et σ de tous les attributs sont stockés séparément. Quand une nouvelle requête arrive, ses valeurs d'attributs ainsi que les poids correspondants sont calculés. Nous avons vu dans la section précédente que les valeurs de poids supérieures à zéro pouvaient correspondre à des requêtes d'attaque et qu'une alarme était déclenchée. Si la requête est réellement une attaque, il s'agit alors d'une véritable détection. Par contre, s'il s'agit d'une requête valide, alors il s'agit d'une fausse alarme et les signatures doivent être mises à jour.

Pour cela, les valeurs de tous les attributs de la nouvelle requête *NR* sont ajoutées aux valeurs précédemment obtenus lors de la phase de paramétrage *R*. Les nouvelles valeurs de moyenne et d'écart type sont calculées en utilisant les équations 1 et 2. Les signatures de ce nouvel ensemble sont alors générées. Ces dernières peuvent alors remplacer les anciennes signatures. Ce processus de mis à jour étant effectué en arrière plan, il n'a pas d'impact sur les performances du traitement des nouvelles requêtes. En outre, étant donné que les signatures sont exprimées sous la forme de vecteurs de bits, le remplacement des anciennes par les nouvelles peut être également effectué sans pénaliser le traitement des nouvelles requêtes.

Fonction `2 Testing_Phase-comp_newrequest`

Data : Ensemble de requêtes de test R.
Result : Alerte pour les requêtes attaques **OU** invalides
begin
compute attributes values for Uri and Query arg.
if $\exists\, Uri$ **then**
compute signature $NR\_signature$ for Uri
$w_{min} \longleftarrow 1$
while $\forall$ $signature\ S \in Uri\ signature\ file$ **do**
$w \longleftarrow NR\_signature\ \&\&\ S$
if $(w/no\_of\_attributes) < w_{min}$ **then**
$w_{min} \longleftarrow w/no\_of\_attributes$
if $w = 0$ **then** break
$w_{max} \longleftarrow 0$; $w_{min} \longleftarrow 1$
if $\exists\, query\_arg$ **then**
compute signature NR_sig for Query arguments
while $\forall$ $query\ arguments$ **do**
while $\forall$ $signature\ S \in query\_arguments\ signature\ file$ **do**
$w \longleftarrow NR\_signature\ \&\&\ S$
if $(w/no\_of\_attributes) < w_{min}$ **then**
$w_{min} \longleftarrow w/no\_of\_attributes$
if $w = 0$ **then** break
if $w_{max} < w_{min}$ **then** $w_{max} \longleftarrow w_{min}$
if w_{min} *of Uri* $\neq 0$ *OR* w_{max} *of Query* $\neq 0$ **then**
alert Attack
if $false\ alarm$ **then**
if $w_{min}\ of\ Uri > 0$ **then**
recomp_signature(U)
if $w_{max}\ of\ Qu\_arguments > 0$ **then**
recomp_signature(Q)
else
$Valid$
end

5 Experimentations

De manière à évaluer la précision de notre SDI pour détecter de nouvelles attaques et le temps nécessaire à la mise à jour des signatures lors de la présence de fausses alarmes, différentes expérimentations ont été menées sur des logs de données réelles issus de la société Beeware et de l'Ecole des Mines d'Alès. Les résultats obtenus étant assez similaires, par manque de place, nous donnons ci-dessous les résultats pour un log composé de **2173831** requêtes valides et **5603** attaques.

5.1 Précision de la détection du SDI

Après avoir appris **1740978** requêtes de l'ensemble de données valides et calculé les signatures, nous obtenons **504497** arguments pour la partie Query et **1740978** Uri pour le jeu d'apprentissage. Le tableau 5.1 décrit la matrice de confusion obtenue pour un test avec **5603** requêtes invalides et **432853** requêtes valides. Ces résultats sont obtenus sans mettre à jour les signatures.

	Valide	**Attaque**
Prédit Valide	432821	*554*
Prédit Attaque	*32*	5044

TAB. 6 – *Matrice de confusion*

La matrice montre que nous avons un taux de vrai détection de requêtes valides de **99.998%** et que le taux de détection d'attaques est de **90.08%**.

5.2 Combien de requêtes doivent être apprises ?

Dans cette expérimentation, nous considérons **23367** requêtes pour l'apprentissage. Mais dans ce cas, nous apprenons une requête et ensuite nous la testons avec **3550** attaques et **7618** requêtes valides. A chaque fois, les signatures sont mises à jour lorsque la nouvelle requête est apprise. Le but de cette expérimentation est d'estimer le nombre de requêtes qu'il est nécessaire d'apprendre avant d'utiliser cette approche sur le serveur. La Figure 1 illustre quelques résultats. Nous pouvons constater que le nombre de fausses alarmes devient constant après un apprentissage de $\approx$ **7500** requêtes. Cette expérience montre qu'en fait il n'est pas obligatoire d'avoir un grand nombre de requêtes pour l'apprentissage. La procédure de mise à jour des signatures nécessite **6.82** secondes après avoir appris **23367** requêtes.

6 Conclusion

Dans cet article, nous avons proposé une nouvelle approche, utilisant les logs générés par les applications Web, pour détecter les tentatives d'intrusion sur un serveur Web. L'approche proposée modélise, à partir d'un ensemble de requêtes valides, les comportements "normaux" sur le serveur en fonction de différents attributs et génère des signatures statistiques utilisées pour détecter les tentatives d'intrusion. Elle ne nécessite pas de spécifier des valeurs de seuil pour les attributs. Elle peut également mettre à jour les signatures pour prendre en compte les évolutions du site. Cette technique a été développée pour détecter les nouvelles attaques plutôt que celles qui sont anciennes ou connues et qui peuvent être détectées par de nombreux SDI. Les expérimentations que nous avons menées ont montré que nous étions capables de reconnaître 99.98% des requêtes valides et plus de 90% de requêtes invalides.

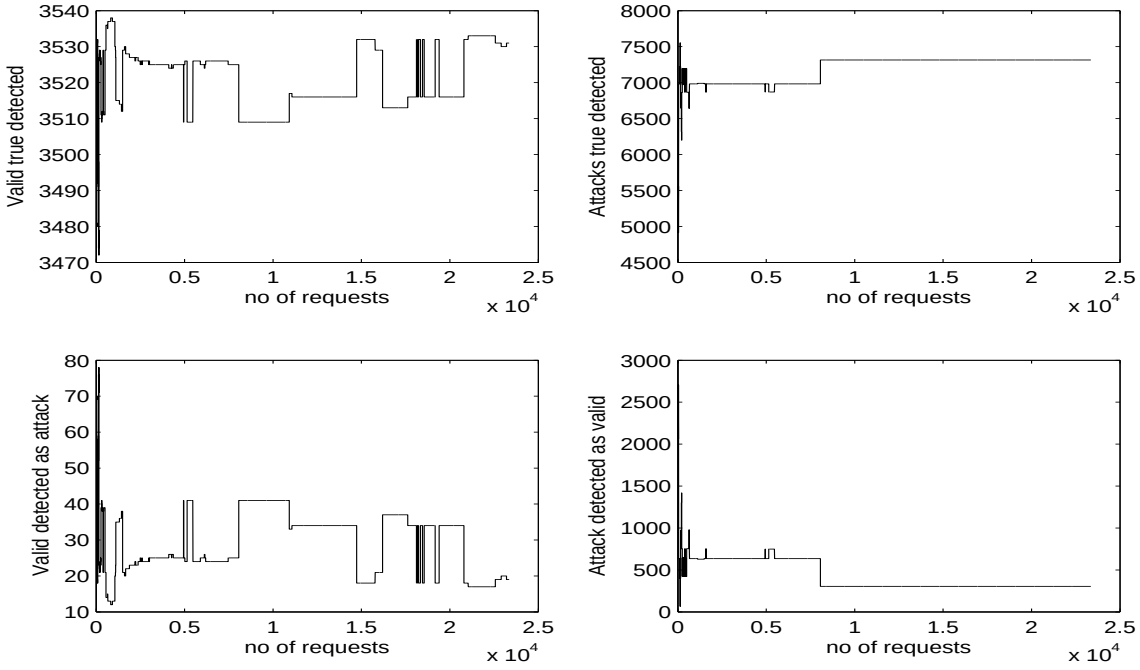

FIG. 1 – *Mise à jour des signatures après apprentissage des requêtes*

Remerciements

Ce travail, réalisé dans le cadre d'un projet de transfert de technologie financé par la région Languedoc Roussillon, résulte d'une collaboration avec l'entreprise BeeWare SA (http ://www.bee-ware.net) qui en exploitera industriellement les résultats.

Références

Adeva, J. J. G. et J. M. P. Atxa (2007). Intrusion detection in web applications using text mining. *Engineering Applications of Artificial Intelligence 20*(4), 555–566.

Apache, D. (2007). http ://httpd.apache.org/docs/.

Berners-Lee, T., L. Masinter, et M. McCahill (1994). Uniform resource locators (url), http ://www.ietf.org/rfc/rfc1738.txt. *http ://www.ietf.org/rfc/rfc1738.txt*.

Cohen, I., F. G. Cozman, N. Sebe, M. C. Cirelo, et T. S. Huang (2004). Semi-supervised learning of classifiers : Theory, algorithms and their application to human-computer interaction.

de Sá Silva, L., A. C. F. dos Santos, T. D. Mancilha, J. D. da Silva Simões, et A. Montes (2007). Detecting attack signatures in the real network with annida. *Elsevier Ltd*.

Denning, D. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering 13*(2), 222–232.

Esposito, M., C. Mazzariello, F. Oliviero, S. Romano, et C. Sansone. Evaluating pattern recognition techniques in intrusion detection systems.

Fogla, P. et W. Lee (2006). Evading network anomaly detection systems : formal reasoning and practical techniques. *Proceedings of the 13th ACM conference on Computer and communications security*, 59–68.

Giacinto, G., R. Perdisci, M. D. Ri, et F. Roli (2006). Intrusion detection in computer networks by a modular ensemble of one-class classifier.

Kruegel, C., G. Vigna, et W. Robertson (2005). A multi-model approach to the detection of web-based attacks. *Computer Networks : The International Journal of Computer and Telecommunications Networking 48*(5), 717–738.

Lee, W., S. J. Stolfo, P. K. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, et J. Zhang (2001). Real time data mining-based intrusion detection. *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings 1*(1), 89–100.

Li, X.-B. (2005). A scalable decision tree system and its application in pattern recognition and intrusion detection. *Decision Support Systems 41*(1), 112–130.

Mangarae, A. et C. Morganti (2007). https ://www.securinfos.info/english/security-papers-hacking-whitepapers.php.

McHugh, J., A. Christie, et J. Allen (2000). Defending yourself : the role of intrusion detection systems. *IEEE Software*, 42–51.

Newsome, J., B. Karp, et D. Song (2005). Polygraph : Automatically generating signatures for polymorphic worms. *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, 226–241.

Paxson, V. (1999). Bro : a system for detecting network intruders in real-time. *Computer Networks 31*(23-24), 2435–2463.

Pereira, A., G. Franco, L. Silva, et W. M. Jr. (2007). A hierarchical characterization of user behavior. *Proceedings of the WebMedia & LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress 00*, 2–9.

Pietraszek, T. et A. Tanner (2005). Data mining and machine learning-towards reducing false positives in intrusion detection. *Elsevier Ltd*.

Proctor, P. (2001). *Practical Intrusion Detection Handbook*. Upper Saddle River, NJ : Prentice-Hall.

Robertson, W., G. Vigna, C. Kruegel, et R. A. Kemmerer (2006). Using generalization and characterization techniques in the anomaly-based detection of web attacks. *Proceedings of Network and Distributed System Security Symposium Conference, Internet Society, 2006*.

Saniee, M., J. Habibi, Z. Barzegar, et M. Sergi (2007). A parallel genetic local search algorithm for intrusion detection in computer networks. *CSICC 2006*.

Wagner, D. et D. Dean (2001). Intrusion detection via static analysis. *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 156.

Yeung, D.-Y. et Y. Ding (2003). Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition 36*(1), 229–243.

Summary

Security of web servers have become a sensitive subject today. Prediction of normal and abnormal requests is hard due to generation of large number of false alarms in many anomaly based IDS. In this paper, we introduce a novel intrusion detection approach using incremental calculation of statistical signatures based on the modelling of normal requests and their distribution value without explicit intervention. Experiments conducted on real datasets have shown high accuracy up to 99.98% for predicting valid request as valid and false positive rate as low as 9.88%.