Génération automatique de signatures par apprentissage pour les systèmes de détection d'intrusions

Omessaad Hamdi, Mbaye Maissa, Francine Krief

Génération automatique de signatures par apprentissage pour les systèmes de détection d'intrusions

Omessaad HAMDI
Université de Bordeaux
LaBRI, France
ohamdi@labri.fr

Maïssa MBAYE
Université Gaston Berger,
LANI, Sénégal
maissa.mbaye@ugb.edu.sn

Francine KRIEF
Université de Bordeaux
LaBRI, France
krief@labri.fr

***Résumé*—Ce papier présente une approche pour automatiser la génération de signatures d'attaques réseaux pour les systèmes de détection d'intrusions. Cette approche est basée sur la programmation logique inductive (apprentissage inductif de programmes logiques) et utilise comme données en entrée une base de traces. Ces traces sont constituées de paquets capturés et classés selon qu'ils appartiennent ou non à une attaque réseau. Le système produit en sortie un ensemble de règles (la signature) décrivant l'attaque. Des simulations préliminaires sur quelques attaques montrent une très bonne précision des signatures apprises par notre approche par rapport à celles disponibles sur la base de l'IDS SNORT.**

I. Introduction

Pour améliorer la sécurité des réseaux, les administrateurs disposent de nombreux outils, dont les systèmes de détection d'intrusions (Intrusion Detection System ou IDS en anglais). Ces outils ont connu un essor particulier et un déploiement massif au cours des dernières années. Les IDS surveillent le trafic et génèrent des alertes lorsqu'il y a des activités malveillantes. Les IDS peuvent être classés en trois catégories : les IDS à base de signatures, les IDS à base d'anomalies et les systèmes hybrides. Les IDS à base d'anomalies construisent un modèle de l'activité normale et tout ce qui s'en écarte est considéré comme une intrusion (anormal). Ce type d'IDS est susceptible de générer beaucoup de faux positifs du fait que tout trafic inconnu est considéré comme une attaque.

Les systèmes à base de signatures se servent de bases de signatures des attaques pour détecter des activités anormales. Cependant, ce mode de fonctionnement les rend très sensibles aux vulnérabilités inconnues (Zero-Day Attack) du système car ils ne gèrent que les attaques dont les signatures se trouvent déjà dans leurs bases de signatures. Par conséquent, cette base doit être alimentée en permanence par les signatures de nouvelles attaques pour être efficace. Un autre inconvénient des systèmes à base de signatures réside dans la difficulté d'établir les signatures une fois qu'un trafic est identifié comme étant une attaque. Malgré, tous ces inconvénients, les IDS à base de signatures sont très performants si l'on considère les deux aspects suivants :

- ils génèrent très peu de faux positifs comparé aux autres types d'IDS,
- ils sont très rapides pour tester les trafics

Si un système permettant d'accélérer l'établissement des signatures, il permettrait à ce type d'IDS d'être beaucoup plus performant.

Dans ce papier, nous proposons une approche qui permet d'automatiser la génération des signatures des attaques qui est une des tâches les plus ardues à accomplir pour les experts. Cette approche consiste à se servir de l'intelligence artificielle, plus particulièrement la PLI (Programmation Logique Inductive) pour générer les signatures à partir des traces des paquets qui font partie d'une attaque. Nous nous y intéressons dans le cadre de l'autoprotection qui permet à une entité de se défendre contre des

attaques.

Ce papier est organisé en deux grandes parties :

- La première présente les concepts de base sur lesquels s'appuie notre approche, notamment les IDS et la PLI.
- La seconde partie détaille la solution que nous proposons ainsi que son implémentation et son évaluation dans des cas d'attaques connues.

Nous finissons par une conclusion résumant nos travaux.

II. Concepts de base

A. Systèmes de Détection d'Intrusions

Un système de détection d'intrusions est un outil qui permet de surveiller le trafic d'un réseau afin de détecter les intrusions et déclencher une alerte. Un IDS analyse le réseau en temps réel, il nécessite donc des ressources matérielles et de la bande passante. Il existe deux types d'IDS (7; 8) :

- NIDS (Network based IDS) qui assure la sécurité au niveau réseau.
- HIDS (Host based IDS) qui assure la sécurité d'une machine particulière.

Les IDS sont aussi classés en deux catégories suivant l'approche : Les IDS à base de signatures et les IDS à base d'anomalies (15; 16).

Les IDS par anomalies utilisent un modèle de comportement censé représenter le profil d'activité normale du système. Quand un comportement s'éloigne de ce profil, une alerte se déclenche. Néanmoins, cet éloignement ne signifie pas forcement une intrusion, ce qui peut générer un taux élevé de faux positifs. Un autre inconvénient réside dans l'obligation du temps d'apprentissage du modèle qui dépend fortement de l'environnement alors que les attaques sont reproduites quasiment à l'identique dans tous les environnements.

Les IDS par signatures utilisent une base de signatures. Les signatures sont généralement définies par un administrateur et doivent être mises à jour pour de nouvelles attaques. Les signatures sont généralement associées à des attaques spécifiques. En cas d'alerte, il est alors assez facile d'identifier l'attaque par un test de conformité. Cependant, cette approche présente la limite de ne pouvoir détecter que les attaques connues du système. Ceci rend la tâche de maintenance de la base de signatures très importante bien qu'elle soit ardue. L'automatisation de cette maintenance a fait l'objet de plusieurs travaux (10), (11) et notre travail s'inscrit dans cette dynamique.

Nous nous intéressons plus spécifiquement dans ce papier aux NIDS à base de signatures.

B. Programmation Logique Inductive

La Programmation Logique Inductive (PLI) a été définie comme étant l'intersection entre l'apprentissage et la programmation logique (6). D'une manière générale, l'induction consiste à dériver une croyance générale T à partir des croyances spécifiques E et un programme logique est, de manière simplifiée, un ensemble de règles et de faits. De manière plus formelle, la Programmation Logique Inductive peut se traduire de la façon suivante : si nous considérons un ensemble d'exemples E (positifs et négatifs), des connaissances sur le domaine étudié B et un ensemble d'hypothèses H, les connaissances liées aux hypothèses doivent couvrir les exemples positifs (E_+) et rejeter les exemples négatifs (E_-). D'où la nécessité de valider la relation suivante :

$$B \wedge H \models E$$

- B, H et E sont chacun des programmes logiques.
- B un ensemble conjonctif de clauses (règles).
- E un ensemble d'exemples : $E_+ \wedge E_-$:
  - E_+ : un ensemble conjonctif de clauses ou faits positifs.
  - E_- : un ensemble conjonctif de clauses ou faits négatifs.
- H l'hypothèse à apprendre.

Une autre manière de présenter serait de dire que la PLI permet d'apprendre un programme logique H qui vérifie les éléments de E_+ et ne vérifie par les éléments de E_- sachant la connaissance de base B. Le langage de programmation logique le plus célèbre est Prolog (5) et nous nous basons sur celui ci pour tout le reste du papier.

Il existe divers approches de la PLI (6) : P-Progol, CProgol, FOIL, FORS, Indlog, MIDOS, SRT, Tilde and WARMR.

A première vue, la PLI permet d'apprendre des règles (clauses ou programmes logiques) et les signatures sont des règles, donc intuitivement la PLI constitue une approche intéressante pour apprendre les signatures. Nous montrerons plus formellement cette relation dans les sections suivantes.

III. Travaux connexes

La détection d'intrusions a été introduite par Anderson (9). Elle permet de signaler les intrusions à l'administrateur de sécurité pour qu'il puisse prendre les mesures adéquates afin de remettre le système dans un état sûr.

Un IDS est généralement évalué selon le taux de faux négatifs (les intrusions non détectées) et les faux positifs (fausses alertes) qui doivent être faibles.

Très peu de travaux ont exploité la programmation logique inductive dans les systèmes de détection d'intrusions (12), (13), (14).

Dans (12), les auteurs ont proposé un système de votes entre la PLI, le GP (Genetic Programming) et un modèle statistique pour diagnostiquer les intrusions. Leur approche est purement théorique. Dans (13), un navigateur a été proposé pour faciliter l'analyse de trafic en utilisant la PLI pour accorder à chaque utilisateur un profil dans l'approche de détection par anomalies. Dans (14), les auteurs se sont servis de la PLI uniquement pour assurer les droits d'accès aux ressources. Dans notre cas, nous nous intéressons aux différents types d'attaques réseaux et nous tentons même de prévoir les attaques inconnues à cet instant ci. De plus, notre approche a été validée par des expérimentations qui ont prouvé que notre méthode permet de reproduire les règles développées par la communauté SNORT d'une façon automatique et avec très peu de traces de paquets suspects.

IV. Apprentissage automatique de signatures basé sur la PLI

A. Principe de base

Dans cette partie, nous présontons une formalisation des signatures des attaques. Tout d'abord, nous définissons l'étape du processus de détection d'intrusions à laquelle nous nous intérressons. Ensuite, nous présentons une formalisation des signatures des attaques afin de pouvoir l'exploiter en PLI.

1) Hypothèses: Lorsqu'une nouvelle intrusion inconnue survient dans un IDS à base de signatures, l'intervention de l'être humain devient nécessaire pour mettre à jour la base de signatures. Le traitement classique effectué par l'expert humain peut être représenté par les étapes figurant sur la figure 1. Le processus commence au moment de la constatation de l'attaque qui permet d'avoir une date de référence pour commencer les prospections. Ensuite, il faut assembler les logs qui constituent une base de travail pour retrouver les trafics à l'origine de l'attaque. Le filtrage des logs va permettre de séparer les paquets qui appartiennent à l'attaque des paquets qui n'en font pas partie. Cette classification permet l'étude des propriétés de cet ensemble de paquets pour en déduire les signatures qui alimenteront la base.

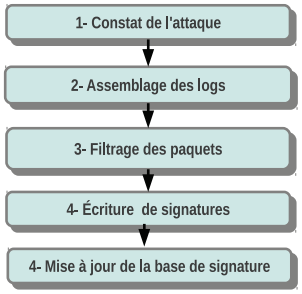

Figure 1. Traitement d'une nouvelle attaque dans un IDS à base de signatures

Cette tâche d'écriture des signatures à partir des paquets est une des tâches les plus difficiles à réaliser pour l'être humain. En effet, plus la quantité de paquets suspects est grande, plus il sera difficile d'établir la signature. Aussi, il est très difficile de trouver les bons paramètres qui permettent de bien décrire l'attaque : la couche, les options, la taille des paquets…

Notre solution intervient à ce niveau pour proposer une approche PLI qui automatise la génération des signatures en l'apprenant automatiquement.

Une hypothèse implicite est donc que les étapes 1 à 3 de la figure 1 sont déjà effectuées en amont

et que notre système ne prend en charge que les paquets filtrés.

2) Propriété clausale des signatures: Considérons un ensemble non vide de paquets $\mathcal{P}$, une condition $\mathcal{C}$ sur les proprietés (champs) des paquets et une fonction $intr(\mathcal{C}, \mathcal{P})$ qui renvoie $VRAI$ si un des paquets de $\mathcal{P}$ vérifie la condition $\mathcal{C}$ et $FAUX$ sinon. Une signature est un ensemble $(\mathcal{C})_{i,j}$ vérifiant la relation :

$$\text{Intrusion} \Leftrightarrow \bigvee_{i=0}^{n} \left(\bigwedge_{j=0}^{n} intr(\mathcal{C}_{i,j}, P) \right)$$

En d'autres termes, une signature dans un IDS est une disjonction de conjonctions sur laquelle sont appliquées des tests de conformités. Si nous développons la disjonction, nous obtenons :

$$Intruision \Leftrightarrow \begin{cases} \bigwedge_{j=0}^{n} intr(\mathcal{C}_{1,j}, P)) & \text{ou} \\ \bigwedge_{j=0}^{n} intr(\mathcal{C}_{2,j}, P)) & \text{ou} \\ \dots & \\ \bigwedge_{j=0}^{n} intr(\mathcal{C}_{n,j}, P)) & \end{cases}$$

Les formes que nous obtenons sont des clauses. L'exercice de formalisation permet de montrer que les signatures ne sont rien d'autres que des programmes logiques. Cela nous permet de justifier l'utilisation de l'apprentissage de programmes logiques représentant les signatures. Il reste maintenant à proposer une modélisation des attaques selon le paradigme PLI.

3) Modelisation PLI: Le principe de notre approche est basé sur l'idée qu'un expert réseau, pour produire la signature d'une attaque réseau, se base sur les logs au moment de l'attaque. Ces logs représentent généralement un ensemble de traces limitées dans le temps et comprenant des paquets qui appartiennent à l'attaque et d'autre qui n'en font pas partie. Nous proposons d'utiliser directement les traces après un effort de classification en deux classes : les paquets qui appartiennent à l'attaque en question et les paquets qui appartiennent au trafic normal (voir figure 2).

Dans notre approche les paquets qui sont les données en entrée sont modélisés comme étant des

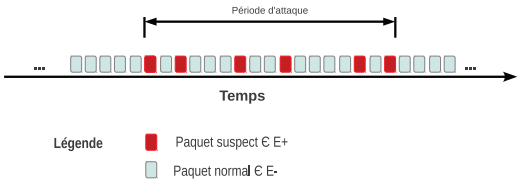

FIGURE 2. Concept de classification des paquets

prédicats dont les paramètres sont les valeurs des propriétés des paquets. De manière plus formelle, la grammaire d'un paquet est la suivante [1] :

`paquet(attr+).`

Où, `attr+` peut être, la longueur du paquet, le protocole de niveau i, ...Ces paquets sont utilisés pour la construction de la base des exemples positifs et négatifs qui ont la forme suivante :

`attaque_descripton(paquet(attr+)`$^+$`)`

Où, `attaque_descripton` est un prédicat qui permet de spécifier qu'un paquet appartient ou non à une attaque, de spécifier $E = E^+ \cup E^-$.

La connaissance de base (Background knowledge), dans cette approche, contient des prédicats pour l'ensemble des propriétés utilisables permettant de décrire la signature d'une attaque réseau. La définition du contenu de cette base dans la pratique a été largement inspirée de la grammaire des règles SNORT [2]. Comme exemples d'éléments de la connaissance de base, nous pouvons citer : `fragoffset, tos, dsize, ip_proto, itype.`

En résumé, l'idée de l'apprentissage de signature utilisant la PLI consiste à se servir des prédicats dans la connaissance de base (B) pour trouver une clause (H représentant la signature), une ou plusieurs règles, qui décrit les propriétés communes à tous les paquets appartenant à l'attaque (E^+) mais qu'on ne retrouve pas dans les autres paquets (E^-).

4) Transformation des signatures: Les signatures apprises sont sous forme de clause Prolog donc la traduction vers d'autres grammaires, comme les règles SNORT, peut être automatisée. Des exemples de signatures apprises sont donnés dans la

1. Sur cette formule c'est plutôt la grammaire que la forme de la logique des prédicats pour éviter de lister tous les arguments
2. http ://manual.snort.org

section suivante.

B. Application et évaluation

1) Environnement d'implémentation: Il existe plusieurs systèmes de détection d'intrusions à base de signatures sur le marché. Nous avons choisi SNORT (4) pour notre implémentation car c'est le plus connu du monde opensource. Il offre des fonctionnalités de traitement en temps réel pour l'analyse de trafic et le logging de paquets IP. Il peut effectuer des filtrages au niveau protocole, de la recherche de contenu et peut détecter une grande variété d'attaques.

Une règle de signature SNORT est formée de deux parties :

- L'entête de la règle qui contient comme informations l'action de la règle, le protocole, les adresses IP source et destination ainsi que les masques réseau, et les ports source et destination.
- Les options de la règle qui englobent les messages d'alerte et les informations sur les parties du paquet qui doivent être inspectées pour déterminer si l'action de la règle doit être acceptée.

Nous avons utilisé SNORT pour enregistrer le trafic qui sera ensuite analysé. Nous avons choisi pour valider les résultats de l'apprentissage de prendre comme référentiel les signatures présentés dans la base de SNORT. En d'autres termes, si nous arrivons à modéliser l'environnement PLI pour retrouver une règle au moins équivalente à celle de SNORT, nous considérerons cela comme étant un succès.

Coté PLI, le logiciel Aleph (A Learning Engine for Proposing Hypotheses) (1), qui est une implémentation basée sur Prolog et qui intègre les approches citées plus haut, a été choisi pour nos implémentations.

Nous nous avons simulé notre approche pour déterminer le nombre d'exemples positifs et négatifs nécessaires pour générer une règle SNORT déjà existante.

La première étape de notre travail consiste à déterminer les caractéristiques d'un paquet réseau. Ces caractéristiques sont ensuite implémentées sous forme de prédicats en Prolog dans la base de connaissance B.

La deuxième étape consiste à simuler un réseau informatique formé d'un serveur et d'un ensemble de machines virtuelles dont une machine attaquante. La simulation a été effectuée en se servant de Netkit (2).

Netkit est un environnement utilisé pour mettre en place et tester un réseau. Il permet de créer plusieurs périphériques réseau virtuels tels que des routeurs, commutateurs, ordinateurs, etc ... et qui peuvent être facilement interconnectés pour former un réseau sur une seule machine.

Les machines envoient des paquets au serveur qui seront analysés et classés en des exemples positifs et des exemples négatifs pour pouvoir ensuite générer une règle logique.

La réalisation des requêtes a été effectuée en se servant de l'utilitaire hping (3), un utilitaire gratuit qui permet de remplacer la commande ping en rajoutant plusieurs fonctionnalités, notamment l'envoi des requêtes TCP ou UDP.

Nous avons simulé plusieurs attaques afin de générer leurs signatures. Nos simulations consistent à étudier le nombre d'exemples positifs et négatifs nécessaires afin de générer une règle logique. Le pourcentage de prédicats exprime le nombre de prédicats apparus dans la règle. Nous présentons, par la suite, nos résultats de simulation pour deux types d'attaques.

2) Application à l'attaque de type SYN flood : Habituellement, dans une connexion TCP client/serveur, le client envoie une requête SYN, le serveur répond alors par un paquet SYN/ACK et ensuite le client valide la connexion par un paquet ACK. Une connexion TCP ne peut s'établir que lorsque ces trois étapes ont été franchies. L'attaque SYN se base sur l'envoi massif de demandes d'ouverture de session SYN. L'attaquant a pour but de saturer le nombre maximum de sessions TCP en cours. Ainsi, la cible ne pourra plus établir aucune autre session TCP.

La règle logique générée par Aleph en se servant du fichier de connaissance que nous avons formalisé et des exemples positifs (paquets issus de l'attaque

Synflood) et négatifs (autre trafic) est la suivante :
paquet_att(A) :- syn_tcp_actif(A), nombre_paquet(A).
qui signifie que la machine a reçu un grand nombre de paquets ayant un flag TCP "SYN" positionné. Notons que dans les simulations, nous avons tenu compte du fait que les paquets attaquants sont issus de la même adresse. Pour cette attaque, un exemple (positif ou négatif) est une séquence de paquets capturés pendant un laps de temps et provenant de la même machine.

Nombre d'$e+$	Nombre d'$e-$	Pourcentage
2	3	100 %
2	2	0 % ou 50%
1	3	0 %
2	1	50 %
0	1	0 %
1	0	0 %

FIGURE 3. Attaque de type SYN flood

La figure 3 présente le nombre d'exemples positifs et négatifs nécessaires pour générer une attaque de type SYN flood avec :

- Nombre d'$e+$ représente le nombre d'exemples positifs ;
- Nombre d'$e-$ représente le nombre d'exemples négatifs ;
- Pourcentage représente le pourcentage de prédicats obtenus par rapport aux prédicats formant la règle logique générée.

Le cas où nous avons 50 % signifie que nous avons obtenu un grand nombre de paquets issus de la même machine où bien des paquets avec SYN actif, ce qui ne signale en aucun cas une attaque de type Synflood.

Remarquons bien que pour pouvoir générer une règle logique équivalente à cette règle définie par la communauté SNORT : *alert tcp any any -> any any (msg :"Syn Flood" ; flow : stateless ; flags :S,12 ; threshold : type threshold, track by_sr(@ip, protocol ; flags tcp , flag ip, port)c, count 3, seconds 1 ; classtype :attempted-recon ; sid :10002 ;rev1 ;)*,
il nous a fallu uniquement 2 exemples positifs et 3 exemples négatifs.

3) Application l'attaque Ping of death: Cette attaque consiste à envoyer un paquet ICMP reply avec une quantité d'informations supérieure à la taille maximale d'un paquet IP (soit 65535). Ce qui peut provoquer un crash de la machine cible.

Nombre d'$e+$	Nombre d'$e-$	Pourcentage
2	4	100 %
2	3	100%
2	2	33% ou 66 %
3	2	33% ou 66%
3	1	33%
2	1	33 %
1	2	0 %

FIGURE 4. Attaque de ping of death

La figure 4 présente le nombre d'exemples positifs et négatifs nécessaires pour générer une règle logique bloquant une attaque de ping of death. Dans cette attaque, un exemple (positif ou négatif) est un paquet. La règle obtenue est :
paquet_att(A) :- dsize(A,65535), icode(A,netunreachable), itype(A,echoReply)
qui signifie un paquet de taille supérieure à 65535 dans un message icmp reply.

Le pourcentage égal à 33% signifie que nous avons obtenu un seul prédicat dans la règle Aleph générée et 66% signifie que nous avons obtenu deux prédicats. L'obtention d'un prédicat dépend des traces captées.

Nous avons simulé d'autres attaques. Les résultats obtenus prouvent qu'avec très peu de traces de paquets suspects, notre approche permet de générer une règle qui décrit et bloque cette intrusion.

L'avantage majeur lié à notre approche, réside dans la capacité d'apprendre au système à identifier les instances d'un comportement anormal d'une manière autonome. En effet, après très peu de paquets suspects reçus, le système est capable de produire une signature liée à cette attaque qui servira ensuite à générer une alerte.

Un autre avantage lié à l'utilisation de la PLI consiste à pouvoir générer des signatures pour des attaques qui ne sont pas encore connues. En effet, avec une connaissance de base bien formulée modélisant les standards du trafic, le système n'aura besoin que de quelques traces de paquets suspects pour générer une signature.

V. Conclusion et Perspectives

Dans ce papier, nous avons présenté une nouvelle approche pour automatiser la génération de signatures pour les systèmes de détection d'intrusions à base de signatures. Notre approche se base sur la programmation logique inductive pour spécifier le comportement d'un réseau en se basant sur la logique du premier ordre. Les signatures des attaques sont générées d'une manière automatique.

En se servant d'une base de connaissance riche et complète définissant un paquet réseau et des exemples obtenus d'un analyseur de réseaux, nous avons réussi à générer des règles logiques similaires à celles de SNORT qui sont conçues et formalisées par des administrateurs réseaux. Notre approche a été testée pour des attaques de type Déni de Service. Ces simulations ont mis en évidence les règles générées par Aleph. En effet, en partant d'une définition d'une attaque, nous sommes capables de reproduire une règle SNORT qui a été généré par un raisonnement humain.

Dans nos simulations, la séparation des exemples positifs et négatifs est effectuée par un administrateur réseaux. Nous travaillons actuellement sur l'automatisation de cette phase du processus. Nous nous intéressons, en particuliers, aux algorithmes de clustering afin d'automatiser la classification des exemples.

Références

[1] Ashwin Srinivasan. The Aleph Manual. http ://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/, Last update : June, 30 2004

[2] http ://wiki.netkit.org. Last update : 19 May 2011 .

[3] http ://www.hping.org/

[4] http ://www.snort.org/

[5] Clocksin, William F. ; Mellish, Christopher S. (2003). Programming in Prolog. Berlin ; New York : Springer-Verlag. ISBN 978-3-540-00678-7.

[6] Lavrac et S.Dzeroski. Inductive logic programming : Techniques and applications. Ellis Horwood, New York, 1994. 120

[7] Anderson, D., Frivold, T.Valdes. A Next-generation Intrusion Detection Expert System (NIDES) : A Summary. SRI International Technical Report SRI-CSL-95-07.

[8] Helman, P., Liepins, G., and Richards, W. (1992). Foundations of Intrusion Detection. In Proceedings of the Fifth Computer Security Foundations Workshop pp. 114-120.

[9] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, April 1980.

[10] David Brumley, James Newsome, Dawn Song, Hao Wang, and So- mesh Jha. Towards automatic generation of vulnerability-based signatures. In SP '06 : Proceedings of the 2006 IEEE Symposium on Security and Privacy, pages 2–16, Washington, DC, USA, 2006. IEEE Computer Society.

[11] James Newsome and Dawn Song. Dynamic taint analysis for au- tomatic detection, analysis, and signature generation of exploits on commodity software. In Proceedings of the 12th Annual Net- work and Distributed System Security Symposium (NDSS 2005), San Diego, CA, February 2005.

[12] Wei Lu. A novel framework for network intrusion detection using learning techniques , Communications, Computers and signal Processing, 2005. PACRIM.

[13] Mizoguchi, F. Anomaly detection using visualization and machine learning, Enabling Technologies : Infrastructure for Collaborative Enterprises, 2000. (WET ICE 2000). Proeedings.

[14] Ko, C. Logic induction of valid behavior specifications for intrusion detection, IEEE Symposium on Security and Privacy, 2000. S & P 2000. Proceedings.

[15] Stefan Axelsson. Intrusion Detection Systems : A Survey and Taxonomy, March 2000

[16] Aleksandar Lazarevic2, Vipin Kumar2 and Jaideep Srivastava, Intrusion Detection : A Survey, Springer US, 2005