

**PRIVACY-PRESERVING INTRUSION DETECTION OVER NETWORK DATA**

by  
LEYLI KARAÇAY

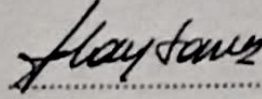
Submitted to the Graduate School of Engineering and Natural Sciences  
in partial fulfilment of  
the requirements for the degree of Doctor of Philosophy

Sabanci University  
December 2019

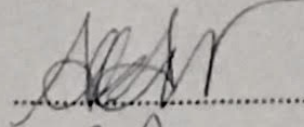
Privacy-Preserving Intrusion Detection over Network Data

APPROVED BY

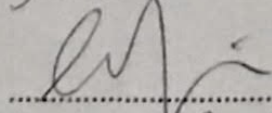
Prof. ErKay Savař  
(Thesis Advisor)



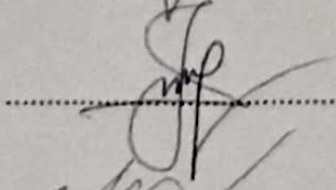
Prof. Albert Levi



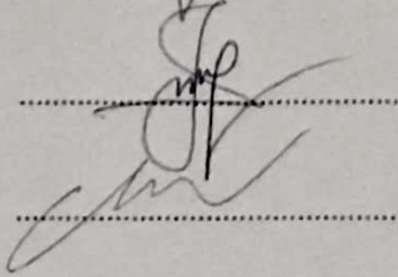
Prof. Cem Güneri



Prof. İbrahim Soğukpınar



Asst. Prof. Uraz Cengiz Türker



DATE OF APPROVAL: December 9, 2019

Privacy-Preserving Intrusion Detection over Network Data

APPROVED BY

Prof. Erkay Savaş .....  
(Thesis Advisor)

Prof. Albert Levi .....

Prof. Cem Güneri .....

Prof. İbrahim Soğukpınar .....

Asst. Prof. Uraz Cengiz Türker .....

DATE OF APPROVAL: December 9, 2019

# Abstract

Privacy-Preserving Intrusion Detection over Network Data

Leyli Karaçay

Ph.D. Dissertation, December 2019

Thesis Advisor: Prof. Erkey Savaş

**Keywords:** *Cyber Security; Intrusion Detection Systems; Lattice-based Homomorphic Encryption;*

Effective protection against cyber-attacks requires constant monitoring and analysis of *system data* such as log files and network packets in an IT infrastructure, which may contain sensitive information. To this end, security operation centers (SOC) are established to detect, analyze, and respond to cyber-security incidents. Security officers at SOC are not necessarily trusted with handling the content of the sensitive and private information, especially in case when SOC services are outsourced as maintaining in-house expertise and capability in cyber-security is expensive. Therefore, an end-to-end security solution is needed for the system data. SOC often utilizes *detection models* either for known types of attacks or for an anomaly and applies them to the collected data to detect cyber-security incidents. The models are usually constructed from historical data

that contains records pertaining to attacks and normal functioning of the IT infrastructure under monitoring; e.g., using machine learning techniques. SOC is also motivated to keep its models confidential for three reasons: i) to capitalize on the models that are its propriety expertise, ii) to protect its detection strategies against adversarial machine learning, in which intelligent and adaptive adversaries carefully manipulate their attack strategy to avoid detection, and iii) the model might have been trained on sensitive information, whereby revealing the model can violate certain laws and regulations. Therefore, detection models are also private. In this dissertation, we propose a scenario in which privacy of both system data and detection models is protected and information leakage is either prevented altogether or quantifiably decreased. Our main approach is to provide an end-to-end encryption for system data and detection models utilizing lattice-based cryptography that allows homomorphic operations over the encrypted data. Assuming that the detection models are previously obtained from training data by SOC, we apply the models to system data homomorphically, whereby the model is encrypted. We take advantage of three different machine learning algorithms to extract intrusion models by training historical data. Using different data sets (two recent data sets, and one outdated but widely used in the intrusion detection literature), the performance of each algorithm is evaluated via the following metrics: i) the time that takes to extract the rules, ii) the time that takes to apply the rules on data homomorphically, iii) the accuracy of the rules in detecting intrusions, and iv) the number of rules. Our experiments demonstrates that the proposed privacy-preserving intrusion detection system (IDS) is feasible in terms of execution times and reliable in terms of accuracy.

# Özet

Ağ Verileri Üzerinden Gizlilik Korunmalı İzinsiz Giriş Algılama

Leyli Karaçay

Doktora Tezi, Aralık 2019

Tez Danışmanı: Prof. ErKay Savaş

**Anahtar Kelimeler:** *Siber Güvenlik; Saldırı Tespit Sistemleri; Kafes Tabanlı Homomorfik Şifreleme;*

Bir BT siber altyapısında saldırılara karşı etkin koruma, günlük bilgileri ve ağ paketleri gibi (hassas bilgiler de içerebilecek) sistem verilerinin sürekli olarak izlenmesini ve analiz edilmesini gerektirir. Bu amaçla, siber güvenlik olaylarını tespit, analiz ve bunlara müdahale etmek için güvenlik operasyon merkezleri (GOM) kurulmuştur. Kuruluşların içerisinde siber güvenlik uzmanlığı ve yeteneğini oluşturmak ve sürdürmek pahalı bir seçim olduğundan sıklıkla GOM hizmetlerinin dışalımına gidilir. Ancak, GOM’nde bu amaçla görevlendirilmiş siber-güvenlik uzmanlarının, siber güvenlik amacıyla işlenen hassas ve özel bilgilere doğrudan erişimi mahremiyet sorunları yaratacaktır. Bu nedenle, sistem verileri için uçtan uca bir güvenlik çözümü gereklidir. GOM genellikle bilinen saldırı türleri veya anomali oluşturan durumlar için saldırı modelleri kullanır ve bunları siber güvenlik olaylarını tespit etmek için toplanan verilere uygular. Modeller genellikle saldırılara ve kayıt altındaki BT altyapısının normal işleyişine ilişkin kayıtları içeren

geçmiş verilerin - örneğin, makine öğrenme tekniklerini kullanarak - işlenmesi sonucunda elde edilirler. Aşağıda verilen üç neden GOM modellerinin gizli tutulmasındaki motivasyonu oluşturur: i) Uzmanlık gerektiren bu modellerden ticari fayda sağlamak, ii) akıllı saldırganların adaptif yöntemler kullanarak bu modellerin kullanıldığı saldırı tespit sistemlerini yanıltmasını önlemek ve iii) model hassas bilgiler kullanılarak eğitilmiş olabileceğinden, modelin ortaya çıkması sonucu belirli yasaların ve düzenlemelerin ihlal edilmesini önlemek. Bu nedenle, saldırı modellerinin de hassas ve mahrem olduğu kabulü yapılır. Bu tezde, hem sistem verilerinin hem de saldırı modellerinin gizliliğinin korunduğu ve bilgi sızıntısının tamamen önlendiği veya ölçülebilir şekilde azaltıldığı bir saldırı tespit senaryosu öneriyoruz. Ana yaklaşımımız, şifrelenmiş veriler üzerinde homomorfik işlemlere izin veren kafes tabanlı şifreleme sistemleri kullanarak, sistem verileri ve saldırı tespit modelleri için uçtan uca şifreleme sağlamaktır. Saldırı tespit modellerinin daha önce GOM tarafından eğitim verilerinden elde edildiğini varsayarak, modellerin şifrelenerek sistem verilerine homomorfik olarak uygulanmasının mümkün olduğunu gösteriyoruz. Verileri eğitmek ve saldırı tespit kurallarını verilerden çıkarmak için üç farklı makine öğrenme algoritmasından yararlanıyoruz. Farklı veri kümeleri kullanılarak, kullandığımız algoritmaların başarımını ölçmek için şu metrikleri kullanıyoruz: i) kuralların çıkarılması için gerekli süre, ii) kuralların homomorfik olarak uygulanması için gerekli süre, iii) siber saldırıları saptamadaki kuralların doğruluğu ve iv) saldırı tespit kurallarının sayısı. Deneylerimiz, önerilen gizliliği ve mahremiyeti koruyan saldırı tespit sisteminin (IDS) çalışma süreleri açısından uygulanabilir ve doğruluk açısından güvenilir olduğunu göstermektedir.

# Acknowledgments

I wish to express my sincere gratitude to my dissertation advisor, Prof. ErKay Savaş, for his continuous support, worthwhile guidance and invaluable patience throughout my graduate studies. I am grateful for all the opportunities that he has provided me. It has been a privilege to study under his guidance. I would also like to thank my dissertation committee members, Prof. Albert Levi, and Prof. Cem Güneri, for their useful feedback and valuable contributions. I am also indebted to the other members of my thesis jury, Prof. İbrahim Soğukpınar and Asst. Prof. Uraz Cengiz Türker, for reviewing my dissertation and providing valuable suggestions and inquiries.

Special thanks to my friend Asst. Prof. Cengiz Örencik who always supported me with his valuable companionship, and unfortunately passed away last year. Also, special thanks to all my friends from Sabanci University Cryptography and Information Security Lab (FENS 2001 and FENS 2014) and all my present colleagues from Ericsson Research company for the great environment they provided. They always supported me with their valuable companionship; we have been like a crowded family.



I am immensely thankful to my family, my mother Mahnaz Attari Jabbarzadeh, for being there when I needed her to be, for believing in me and for supporting me throughout all my decisions. I would not be here without her unlimited love and support. I cannot find words to express my appreciation for my mother. I also need to thank to my mother in law, Mansoureh Bagheri Darbandi, for supporting me in every aspect. I feel very lucky to have such parents.

Last but definitely not the least, I am beyond grateful for the presence of my husband, Aydın Karaçay. He has walked through the journey together with me and shared the new horizon over these years. When I needed motivation the most, his unlimited moral, continual support and patience aided me. He was always around at times I thought that it is impossible to continue and he helped me to keep things in perspective. I greatly value his contribution and deeply appreciate his belief in me. I feel very lucky to have him and his endless unconditional love. Finally, Alp Karaçay and Aran Karaçay, our beloved sons, I'm extremely happy to have them. I owe them a debt of gratitude ahead of time because of their well-behaved, endless love, and not giving any discomfort to me. Words would never say how grateful I am to all of them.

I consider myself the luckiest person in the world to have such a lovely and caring family, standing beside me with their love and unconditional support.

# Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>2</b>
1.1	Contribution . . . . .	3
1.2	Preliminaries and Notation . . . . .	4
1.2.1	Definitions . . . . .	4
1.2.2	Problem Setting . . . . .	7
1.2.3	Notation . . . . .	8
1.3	Outline . . . . .	8
<b>2</b>	<b>BACKGROUND INFORMATION</b>	<b>10</b>
2.1	Intrusion Detection Systems (IDS) . . . . .	10
2.1.1	Detection Method of IDS . . . . .	12
2.2	Homomorphic Encryption . . . . .	12
2.3	Lattice-based Cryptography . . . . .	14
2.4	Machine Learning Algorithms . . . . .	14
2.4.1	Decision Tree Algorithm . . . . .	15
2.4.2	Naïve Bayesian Algorithm . . . . .	15
2.4.3	Neural Network Algorithm . . . . .	16
2.5	Evaluation of Classification Algorithms . . . . .	16
2.5.1	Evaluation Metrics . . . . .	17
2.5.2	Precision-Recall Curves . . . . .	17
2.5.3	Confusion matrix . . . . .	18
2.6	Data Pre-processing Techniques . . . . .	18
2.6.1	Attribute selection . . . . .	18
<b>3</b>	<b>RELATED WORK</b>	<b>21</b>
3.1	Literature overview on data sets used in NIDS . . . . .	21
3.2	Literature overview on rule-based machine learning . . . . .	23
3.3	Literature overview on similar works . . . . .	24
<b>4</b>	<b>Rule-based Classification Techniques</b>	<b>26</b>
4.1	Decision Tree-based Model . . . . .	27

4.2	Naïve Bayesian-based Model . . . . .	28
4.3	Neural Network-based Model . . . . .	33
4.3.1	Learning Rules . . . . .	33
4.3.2	Classification . . . . .	35
4.3.3	Rule-based Intrusion Detection Using Rules from Neural Network	37
<b>5</b>	<b>Privacy-Preserving Intrusion Detection</b>	<b>40</b>
5.1	Participants . . . . .	40
5.2	Semi-honest Protocol . . . . .	40
5.3	Proposed Construction . . . . .	42
5.3.1	Record Signature Generation . . . . .	43
5.3.2	Rule Signature Generation . . . . .	43
5.3.3	Intrusion Detection Algorithm . . . . .	44
<b>6</b>	<b>Privacy and Security Arguments</b>	<b>48</b>
<b>7</b>	<b>Experiments</b>	<b>54</b>
7.1	Experimental Setup . . . . .	54
7.1.1	Feature Selection . . . . .	55
7.2	Evaluation Metrics . . . . .	55
7.3	Results and Discussions . . . . .	56
7.3.1	Decision-tree based model results . . . . .	57
7.3.2	Performance Comparison of Three Classification Methods . . . . .	64
<b>8</b>	<b>Conclusion</b>	<b>70</b>

# List of Figures

1.1	Binary decision tree as a detection model. . . . .	5
2.1	Intrusion Detection System in the network environment . . . . .	11
2.2	Confusion matrix . . . . .	18
4.1	Binary decision tree as a detection model. . . . .	27
4.2	General architecture of rule-based neural network . . . . .	36
4.3	A simple architecture of the rule-based neural network. . . . .	37
5.1	Block diagram of the overall scheme. . . . .	41
7.1	Variation of the number of attack rules ( $u$ ) with respect to the decision tree depth in CIC-Friday-Afternoon data set. . . . .	60
7.2	DO and SOC computation time (excluding network communication) for one record in the CIC-Friday-Afternoon data set with respect to the number of attack rules with $t = 25$ and $\ell = 512$ . . . . .	61
7.3	DO and SOC communication uploads for one record in the CIC-Friday-Afternoon data set with respect to the number of attack rules with $t = 25$ and $\ell = 512$ . . . . .	61
7.4	DO computation time for 4064 records in CIC-Friday-Afternoon data set depending on dimension of the feature vector with $t = 25$ and $u = 19$ . . . . .	62
7.5	DO upload for 4064 records in CIC-Friday-Afternoon data set depending on dimension of the feature vector with $t = 25$ and $u = 19$ . . . . .	63
7.6	DO's computation time depending on number of records in CIC-Friday-Afternoon with $\ell = 512$ . . . . .	64
7.7	DO's upload depending on number of records in CIC-Friday-Afternoon with $\ell = 512$ . . . . .	64
7.8	Comparison of 3 rule-based classifier on samples of the CIC-Friday-Morning data set. . . . .	66
7.9	Comparison of 3 rule-based classifier on sample of the CIC-Friday-Afternoon data set . . . . .	66
7.10	Comparison of 3 rule-based classifier on sample of the KDD Cup 1999 data set . . . . .	67

# List of Tables

4.1	CIC-Friday-Morning sample data set’s features and their domain. . . . .	31
4.2	Class probability values for some of the combinations derived from CIC-Friday-Morning . . . . .	32
6.1	Comparison of entropy of numerical variables and discretized variables in ISCX-Saturday data set . . . . .	50
7.1	Parameters used by WEKA : $\text{Attack}(\#)$ is the number of malicious records in the random sample data set, $d$ is the dimension of the feature vector. . .	57
7.2	Parameters used in our protocol: $\mathcal{T}_{IG}$ is the threshold for deleting irrelevant features, $d$ is the dimension of the feature vector after feature elimination, $t$ is the number of categories of the feature with the maximum number of categories. . . . .	58
7.3	Comparison of detection results by WEKA and the proposed protocol over three data sets. . . . .	59
7.4	Parameters: $d$ is the dimension of the feature vector, $d_t$ is the depth of the decision tree model, $d_n$ is the number of decision nodes. . . . .	59
7.5	Data sets characteristics . . . . .	65
7.6	Number of attack rules extracted by each rule-based classifier . . . . .	68
7.7	Rule extraction time (millisecond) . . . . .	68
7.8	Intrusion detection time (millisecond) . . . . .	68

# LIST OF ABBREVIATIONS

IDS	Intrusion Detection System
CADS	Cyber Attack Detection Systems
NIDS	Network Intrusion Detection System
HIDS	Hos Intrusion Detection System
SOC	Security Operational Center
DO	Data Owner
DT	Decision Tree
BDT	Binary Decision Tree
NN	Neural Network
SSO	Site Security Officer
DoS	Denial of Services
DDoS	Distributed Denial of Service
SEAL	Simple Encrypted Arithmetic Library
HE	Homomorphic Encryption
SWHE	Some What Homomorphic Encryption
ROC	Receiver Operating Characteristic
FP	False Positive
FN	False Negative
TP	True Positive
TN	True Negative
CIC	Canadian Institute for Cybersecurity

# Chapter 1

## INTRODUCTION

In recent years, IT infrastructures are becoming increasingly vulnerable to sophisticated forms of cyber-attacks [1]. As defensive tools, *cyber attack detection systems* (CADS) have proved to be reliable in detecting cyber-attacks such as Probe, DoS, U2R, and R2L [1] with low false alarm rates. Most CADS rely on essentially two methods for effective detection: i) monitoring IT infrastructures to collect system data such as network packets and system logs, and ii) detection models (e.g., attack signatures, classifiers, anomaly detection techniques) that are used to classify the system data.

Naturally, accurate detection models play a crucial role in the performance of CADS. Furthermore, sufficiently accurate detection models can only be built through a rich set of historical data pertaining to attacks, high level of expertise in the field, and timely cyber-intelligence data. Also, prevention, mitigation and response after detection require expert teams with certain skill sets and well-defined sets of actions. Thus, outsourcing of CADS to security professionals stands as an effective strategy for many organizations, whose core businesses are not in security. Cloud-based security operation centers (SOC), while being an economical and convenient alternatives, introduce new challenges as far as the privacy of SOC and service users are concerned.

*Privacy of the service users:* CADS detect potential and emerging attacks by monitoring many activities in IT infrastructures consisting of network links and computers. This is carried out by collecting and analyzing system data, which is taken from various sources such as system log files or network traffic and can be used to infer sensitive information about individuals, companies, and organizations [2, 3]. Therefore, processing of sensitive data by external SOC can raise multiple privacy concerns. For example, content of a network packet or information about a connection can reveal significant amount of information, potentially related to a day-to-day operation of a company, which is valuable from a business decision point of view and thus, may well be regarded as sensitive.

*Secrecy of the detection model:* CADS often utilize statistical and machine learning models to detect a behaviour, which is believed to be due to an attack. From a service provider perspective, keeping the underlying model private is crucial for three main rea-

sons. Firstly, a detection model is propriety knowledge that should be protected against competitors (and possibly service users themselves). Secondly, an adversary knowing the model can alter his tactics in such a way that an alarm is not triggered by the model. And lastly, the model itself can leak information about the historical data that have been used to train the model as it can be private and/or sensitive as well.

## 1.1 Contribution

In this dissertation, we propose a practical framework for private evaluation of detection models on network data packets for intrusion detection. In our setting, SOC has an intrusion detection model and client (by owning the data, referred as also data owner (DO) henceforth) holds data to the model. Abstractly, our desired security property is that at the end of the protocol execution, SOC learns nothing about DO’s data, and DO learns nothing about SOC’s model other than what can be directly inferred from the protocol output.

We utilize a lattice-based somewhat homomorphic encryption (SWHE) algorithm to encrypt the model rather than the data, which is in contrast with similar works in the literature, in which usually homomorphically encrypted data is sent to a server for evaluation. Our approach is to run homomorphic evaluation on client side (i.e., in DO’s computers), and the server (SOC) is needed only to decrypt the evaluation result, which is simply the label of a class, to which a particular data item belongs. This can be advantageous as the evaluation is performed closer to where data is produced.

We propose a new set of security definitions for this new setting and provide security analysis as to how the protocol in the proposed framework satisfies those definitions. All previous works in the literature implicitly assume that data attributes and their domain are known to the client, which may leak information about the model. For instance, a most common model is decision tree, where a set of comparison operations are applied successively for classification. Furthermore, even the type of comparison operation is made known such as “greater than or equal to”. While this is required for classification we still need to quantify what is given away when attributes, their domain and comparison operations are known, as a malicious client can send successive queries to server to learn the decision tree from the corresponding outputs. We use Shannon entropy to measure how much is still unknown about the decision tree after they are shared with clients. To this end, a new security definition for “predicate privacy” is introduced. Our analysis based on entropy and predicate privacy allows us to calculate the attack cost of a malicious client in the worst case. Also, we demonstrate that class labels need not be shared with clients in intrusion detection applications; only a specific preventive/responsive action is, when an attack is detected. This will minimize the leakage of the model to the client, which may never be able to learn the exact model, but only an approximation of it. Similarly, leakage



of system data to server is also minimized.

We use very recent and more realistic data sets to verify the accuracy of the proposed privacy-preserving intrusion detection protocol while in the literature old data sets are still being used such as DARPA 1998 by MIT Lincoln Labs [4]. Our experimental results demonstrate that the proposed method leads to no deterioration in accuracy when compared to those by tools such as WEKA. Although we use homomorphic encryption algorithms as our basic security primitive, which is usually considered slow, the performance results are very promising for real world applications and compare favorably with those by works that deal with private evaluation of decision trees.

## 1.2 Preliminaries and Notation

In this section, we provide the definitions, the problem setting, and the notation.

### 1.2.1 Definitions

Here, we give definitions to clarify the terminology used in the rest of the paper.

**Definition 1.1** (Security Data). *Any data that is collected from a networked computer system, which is generated intentionally or unintentionally as a result of system events/activities and utilized to detect anomalies, suspicious behavior, threats, attacks and unauthorized actions, are referred as security data. Examples include system log files, network packets, system calls by applications.*

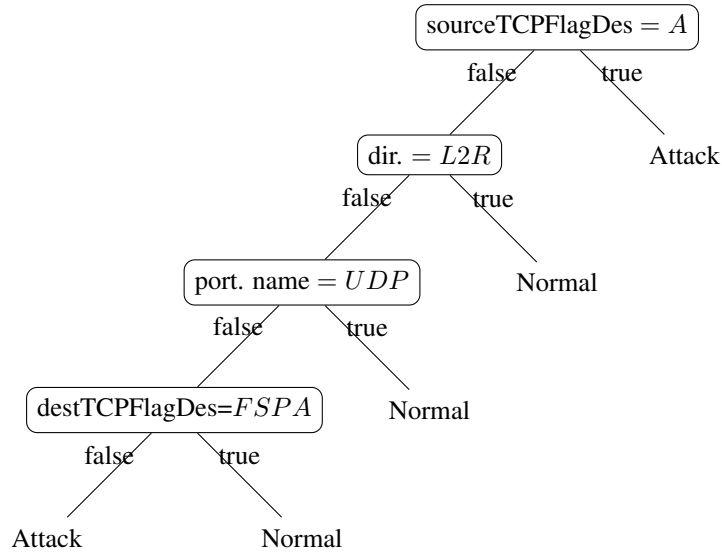
The organization, from whose computer system the security data is collected, is called *data owner* (DO). The security data is considered a private database  $\mathcal{D}$  and consist of records, each of which is associated with a system event; i.e.,  $\mathcal{D} = \{r_1, \dots, r_i, \dots\}$ , where  $r_i$  represents an individual record and record generation is a continuous process.

Let  $\mathcal{A} = \{a_1, \dots, a_d\}$  be the feature set in  $\mathcal{D}$ , and  $\mathcal{V}_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,t}\}$  be the set of all possible values such that  $a_i \in \mathcal{V}_i$ . A record  $r$  is, then, a vector of dimension  $d$ ,  $r = (a_1, \dots, a_d)$ , called a feature vector. Thus, in this paper, we always use feature vectors that contain categorical variables.

**Definition 1.2** (Intrusion Detection System). *An intrusion detection system (IDS) is a class of security software deployed to monitor the security data and generate alert messages when there is an attempt to compromise the security of a system via malicious activities or security policy violations.*

**Definition 1.3** (Detection model). *Detection model is a procedure learned from the historical security data and applied to new data instances for intrusion detection.*

Figure 1.1: Binary decision tree as a detection model.



**Example 1.1.** A binary decision tree is an example of detection model as shown in Figure 4.1. The features of network activities (e.g., direction of the flow, protocol name, source and destination tcp flag descriptions, see [5] for more information) are used in the nodes of the decision tree. The tree can be used to classify the network connections either as an Attack (“A”) or Normal (“N”).

**Definition 1.4 (Predicate).** Let  $op \in \{=, \neq\}$  be an operator on categorical variables. A predicate  $p_i$  is defined as a Boolean expression  $p_i(a_i) \leftarrow (a_i \text{ op } v_{i,j})$ , where  $v_{i,j} \in \mathcal{V}_i$  and  $p_i(a_i) \in \{True, False\}$ .

**Example 1.2.** Suppose features are

- $a_1 = \text{“sourceTCPFlagDescription”}$ ,
- $a_2 = \text{“direction”}$ ,
- $a_3 = \text{“protocol name”}$ ,
- $a_4 = \text{“destinationTCPFlagDescription”}$ ;

and feature values are

$$\begin{aligned}\mathcal{V}_1 &= \{N/A, FA, A, FSPA, SPA, FSRPA, S, SRPA, FSA, \\ &FPA, PA, SA, RA, FRA, R, SR, RPA, FRPA, FPU, \\ &SRIllegal7Illegal8, FSRPU, FSPU, FSRA, SRA\}, \\ \mathcal{V}_2 &= \{L2R, L2L, R2L, R2R\}, \\ \mathcal{V}_3 &= \{TCP, UDP, IP, IGMP, ICMP\}, \\ \mathcal{V}_4 &= \{N/A, R, FA, PA, FSPA, SPA, FSRPA, SRPA, \\ &FRA, A, FPA, SA, FRPA, FSA, FSRA, RA, SRA, \\ &SRAIllegal8, RPA, FSPA, Illegal8\}.\end{aligned}$$

Given the binary decision tree (BDT) in Figure 4.1, each internal node is associated with a Boolean expression and each leaf node is associated with an output value (class labels). At each internal node, depending on whether the Boolean expression evaluates to TRUE or FALSE, either the right or left branch of the tree will be taken. The predicates for the tree are  $p_1 \leftarrow (a_1 = A)$ ,  $p_2 \leftarrow (a_2 = L2R)$ ,  $p_3 \leftarrow (a_3 = UDP)$ , and  $p_4 \leftarrow (a_4 = FSPA)$ .

**Definition 1.5 (Rule).** A rule is a set of conjoined predicates (i.e., combined with logical AND operation) that corresponds to a path from the root node to a leaf node. If the leaf node is in one of the attack classes, then it is known as attack (or intrusion) rule.

**Example 1.3.** In a BDT, a path from the root node to a leaf node defines a rule. There are two rules for the class label “Attack” in Figure 4.1:  $R_1 = p_1$  and  $R_2 = \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4$ .

In fact, a rule can be written in the more general form

$$R_i = \mathbf{IF} \rho_i \mathbf{THEN} x_j,$$

where  $\rho_i$  is the conjunction of the predicates and  $x_j$  is a class label. However, as we are interested in the attack class, we sometimes use  $R_i$  and  $\rho_i$  interchangeably.

**Definition 1.6 (Attack Policy).** Attack policy is the set of rules which are disjointly applied (i.e., using logical OR operation) to reach all leaves labeled in the same attack class.

**Example 1.4.** The attack policy in Figure 4.1 can be given as:

$$P_{Attack} = R_1 \vee R_2, \text{ where } R_1 = p_1 \text{ and } R_2 = \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4.$$

Similarly the policy for normal traffic in Figure 4.1 can be given as:

$$P_{Normal} = R_3 \vee R_4 \vee R_5, \text{ where } R_3 = \neg p_1 \wedge p_2 \text{ and } R_4 = \neg p_1 \wedge \neg p_2 \wedge p_3 \text{ and } R_5 = \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge p_4.$$

As we can have more than one attack type (a separate class), we need to define intrusion policy as well.

**Definition 1.7** (Intrusion Policy). *Intrusion policy is the set of attack policies which are disjointly applied (i.e., using logical OR operation) to reach each leaf labeled in one class of attacks.*

A record in security data that satisfies one or more attack rule is called *offensive record*.

## 1.2.2 Problem Setting

As the intrusion detection is often outsourced, there are mainly two distinct parties in the proposed scenario: Data owner (DO) that holds records of security data and security operation center (SOC) that holds an intrusion policy. A privacy-preserving intrusion detection protocol requires that no information about either the security data or the detection model be leaked to the other party (or any other party) during the private detection protocol except for what can be inferred from the protocol output in the ideal world [6]. Our main point of defense is homomorphic encryption scheme used in an interactive protocol to provide the privacy of records of security data and intrusion policy. Given ciphertexts that encrypt plaintext inputs  $\pi_1, \dots, \pi_t$ , a fully homomorphic encryption (FHE) scheme allows anyone to output a ciphertext that encrypts  $f(\pi_1, \dots, \pi_t)$  for any desired function  $f$ , as long as it can be efficiently computed. No information about  $\pi_1, \dots, \pi_t, f(\pi_1, \dots, \pi_t)$ , or any intermediate plaintext values, leak; the inputs, output and intermediate values are always encrypted [7]. Somewhat fully homomorphic encryption (SWHE) is FHE that supports only limited number of homomorphic operations. As SWHE schemes are much more practical than FHE schemes, we use one such scheme in our work, which will always be referred as the HE scheme henceforth.

Lattice-based HE schemes [8, 9] use two moduli: plaintext modulus  $p$  and ciphertext modulus  $q$ , where  $q \gg p$  and  $p \geq 2$ . This simply means that a ciphertext decrypts into integers in the interval  $[0, p - 1]$  and arithmetic operations performed homomorphically over a ciphertext result in modulo  $p$  arithmetic over the plaintext. Also lattice-based HE schemes enable SIMD (single instruction multiple data) operations over a ciphertext. Simply put, a ciphertext encrypts independent slots, each of which encrypts a modulo- $p$  integer. When a homomorphic operation is applied to a ciphertext, the same operation is independently applied to all slots simultaneously; a property that is referred as *batching*. For instance, suppose two ciphertexts,  $\rho_1$  and  $\rho_2$  encrypt  $k$  slots of integers each; namely  $\rho_1 = E(a_1; \dots; a_k)$  and  $\rho_2 = E(b_1; \dots; b_k)$ , where  $E$  stands for homomorphic encryption function. Then

$$D(\rho_1 + \rho_2) = a_1 + b_1 \pmod{p}; \dots; a_k + b_k \pmod{p},$$

where  $D$  is the decryption function. Other homomorphic operations are also possible such as shift, rotation and permutations of slots, and combining the slots of different ciphertexts into another ciphertext.

Also, in lattice-based HE schemes a ciphertext is a pair of polynomials of degree at most  $N - 1$  and each polynomial is the element of the ring  $R_q = \mathbb{Z}_q[x]/(x^N + 1)$ , where  $x^N + 1$  is a cyclotomic polynomial and  $\mathbb{Z}_q[x]$  is the set of all polynomials whose coefficients are reduced modulo  $q$ . Similarly, each plaintext is encoded before encryption as a single polynomial which is in ring  $R_p = \mathbb{Z}_p[x]/(x^N + 1)$ . We will refer  $N$  as the ring degree henceforth.

In Chapter 6 we provide formal privacy definitions for a privacy-preserving intrusion detection system. The proofs that indicate our proposed scheme is privacy-preserving are also given.

### 1.2.3 Notation

Let  $\mathbf{s}$  be a row vector of length  $\ell$ . Then,  $\mathbf{s} \leftarrow (c)^{1 \times \ell}$  initializes all elements of  $\mathbf{s}$  to the constant value  $c$ . Similarly, let  $\mathbf{M}$  be a matrix of dimension  $u \times \ell$ ; then  $\mathbf{M} \leftarrow (c)^{u \times \ell}$  initializes all elements of  $\mathbf{M}$  to the constant value  $c$ . Alternatively, vectors and rows of binary matrices are also referred as strings. For instance, a binary vector is the same as binary string. Right and left rotations of a string  $\mathbf{s}$  by  $x$  digits are denoted by  $\mathbf{s} \gg x$  and  $\mathbf{s} \ll x$ , respectively.

Boldface lowercase and uppercase letters are used for vectors and matrices, respectively. Normal font letters are used either for scalars or for variables of unknown/unspecified type.

The notation  $\chi \stackrel{\mathbf{R}}{\leftarrow} \mathbb{Z}_p$  designates uniform random sampling from the interval  $[0, p - 1]$ .  $\mathcal{D}[r_j, a_i]$  represents the value of a feature  $a_i$  in a record  $r_j$  in the data set  $\mathcal{D}$ ; i.e.,  $\mathcal{D}[r_j, a_i] \in \mathcal{V}_i$ . Finally,  $p_i(a_i)$ , the evaluation of predicate  $p_i$  on a value of  $a_i$ , returns TRUE or FALSE.

## 1.3 Outline

The organization of this dissertation is as follows: The related background information is provided in Chapter 2, including brief introduction to intrusion detection systems (IDS), homomorphic encryption, lattice-based cryptography, machine learning algorithms, evaluation of classification algorithms, and data pre-processing techniques. In Chapter 3, we review the cryptographic techniques that are already used for intrusion detection systems, together with the literature on rule-based machine learning algorithms. Chapter 4 explains the techniques which are used to extract attack rules from training data using three classification algorithms (i) Decision-tree, (ii) Naïve Bayesian, and (iii) Neural Network.

In Chapter 5 we introduce our novel privacy-preserving intrusion detection protocol, and provide two different algorithms for generating record and rule signature, and also the algorithm for intrusion detection. In Chapter 6 we provide the security requirements and we argue that the proposed protocol addresses those requirements. In Chapter 7 we evaluate the performance of the provided protocol in terms of computation time and bandwidth over three real data sets and we compare our protocol with similar existing works in the literature. In addition, we compare the performance of the three different machine learning algorithms, and explain about advantages and limitations of each algorithm. Finally, Chapter 8 concludes this dissertation and give prospects for future work.

# Chapter 2

## BACKGROUND INFORMATION

This chapter is devoted to providing a preliminary background in intrusion detection systems (IDS), homomorphic encryption, lattice-based cryptography, machine learning algorithms, evaluation of classification algorithms, and data pre-processing techniques.

### 2.1 Intrusion Detection Systems (IDS)

Most of the activities on the internet such as shopping, paying bills, banking etc., involve money exchange and transferring of critical information over the network. Almost exponential growth in the number of applications and the size of computer networks lead to a dramatic increase and seriousness in cyber-attacks which may cause damage from financial costs to personal privacy and national security. The diversity of the attacks and their constantly changing nature hinder timely development of effective countermeasures on a par with the sophistication of attacks. Therefore, security solutions that are capable of analyzing large amounts of network traffic and detecting variety of attacks, are required. Intrusion Detection Systems (IDS) [10] are one of such solutions. In the computer security field, intrusion detection systems attempts to detect intrusions by monitoring suspicious activity and issues alerts when such activity is discovered [11, 12]. In such cases an entity, most often a site security officer (SSO), can respond to the alarm and take appropriate actions. IDS collects information about the system being observed using its audit data collection agent. This data is then either stored or processed directly by the detector, the output of which is presented to SSO, who then can take further actions; normally starting further investigation into the causes of the alarm [13].

IDS is normally classified into two types: i) network based intrusion detection systems (NIDS) and ii) host based intrusion detection systems (HIDS). The brief explanation for each type is provided in the following sections.

## Network-based Intrusion Detection System (NIDS)

NIDS monitors network traffic, and usually is placed at a strategic point within the network to examine traffic to and from all devices in the network [14, 15]. It observes the passing traffic on the entire subnet, and matches the traffic to the collection of known attacks. Once an attack is identified or abnormal behavior is observed, an alert message can be sent to the administrator. The NIDS is usually located at a network device immediately next to a firewall (see Figure 2.1). Additionally, an NIDS is unable to decrypt encrypted traffic. In other words, it can only monitor and estimate threats on the network from traffic sent in plaintext.

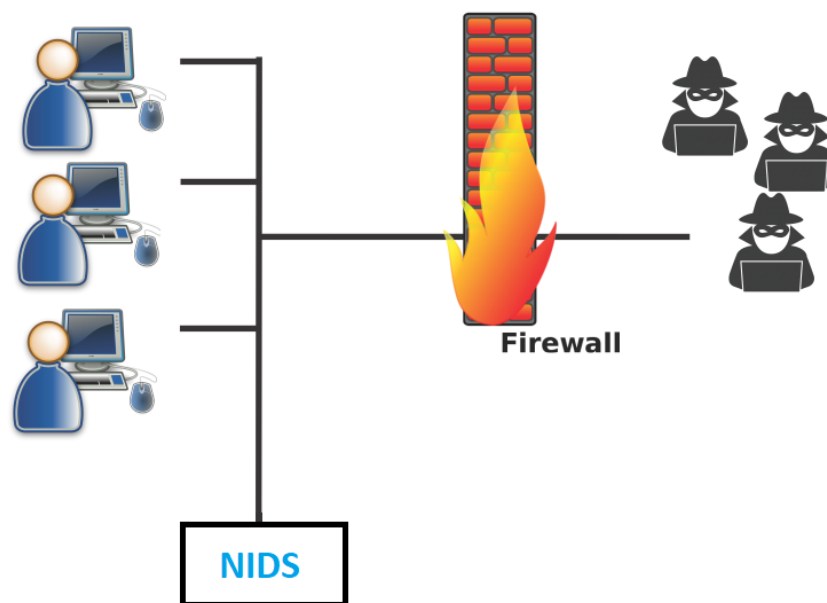


Figure 2.1: Intrusion Detection System in the network environment

## Host-based Intrusion Detection System (HIDS)

Host-based intrusion detection system (HIDS) runs on independent hosts or devices in the network. An HIDS monitors activities such as file changes, system logs, and host based network traffics from the device only and will alert the administrator if suspicious or malicious activity is detected [16]. It takes a snapshot of existing system files and compares it with the previous snapshot. If the analytical system files were edited or deleted, an alert is sent to the administrator to investigate. An example of HIDS usage can be seen on mission critical machines, which are not expected to change their layout.



### 2.1.1 Detection Method of IDS

All IDSs use one of two detection techniques: i) signature-based and ii) anomaly-based. They are explained next.

**Signature-based IDS.** In signature-based IDS, specific knowledge of intrusive behavior [17] is required. All instances that constitute legal or illegal behaviour are investigated to derive specific patterns and signatures. In a signature based IDS, predetermined attack patterns are defined in the form of signatures used to determine the cyber-attacks [18]. In signature-based IDS, intrusive activity is detected based on signatures and no knowledge regarding the normal behaviour of the system is needed.

**Anomaly-based IDS.** Anomaly detection [19] does not search for known intrusions; but rather for abnormalities in the system activity based on the premise of “something that is abnormal is probably suspicious” [13]. Actually, what is “normal” for the system is determined by the IDS from regular system activity (i.e., “baseline”) such as bandwidth usage, protocols that are generally used, or what ports and devices are generally connected to each other. The administrator is alerted when an activity significantly different from the baseline is detected [20]. If the baseline is not configured intelligently, then it may raise a false positive alarm for legitimate activity. Similarly, an ill-formed baseline can lead to false negatives, whereby an attack is not detected

## 2.2 Homomorphic Encryption

Homomorphic encryption (HE) is a form of encryption that enables computation on ciphertexts without access to the secret key or need for decryption; the encrypted result is generated which, when decrypted, matches the result of the operations as if they had been performed on the plaintext. HE is especially useful for privacy-preserving computation over data, whose storage is outsourced to third parties. In particular, after having been homomorphically encrypted, data can be outsourced to a commercial cloud storage service and processed while it is encrypted. HE can be used in highly regulated industries such as health care, to enable new services, where data barriers can be removed by inhibiting data sharing. Sony’s play station network got hacked in 2011 and billions of personal information got disclosed due to unencrypted data storage [21]. Actually, HE is an effective solution for organizations who are seeking to process information while still protecting privacy and security. Unlike other encryption algorithms in use today, lattice-based HE algorithms are claimed to be safe against quantum computer attacks. A public key is used to encrypt the data, and the algebraic structure in lattice-based HE systems is utilized to allow functions to be performed directly on the encrypted data. After applying the functions to the encrypted data, the result can be accessed only by the party that owns the private key.

HE includes different types of encryption schemes that can perform different classes of computations over encrypted data [22]. Widely known types of homomorphic encryption are partially homomorphic encryption (PHE), somewhat homomorphic encryption (SHE), and fully Homomorphic encryption (FHE). The computations are represented as either Boolean or arithmetic circuits.

**Partially Homomorphic Encryption.** PHE [23] supports only some homomorphic operations. For example, addition and multiplication are such operations, where only one of them can be performed on the encrypted data but not both, depending on the particular PHE system.

**Somewhat Homomorphic Encryption.** SWHE [24] supports all sorts of arithmetic and logic operations. The most important drawback of SWHE system is that the number of homomorphic operations is limited. Another limitation of SWHE is that not all operations can be applied to all types of data at the same time. SWHE is suitable for a variety of real time applications such as financial, medical and recommender systems. Since SWHE supports a limited number of operations it will be much faster than fully homomorphic schemes, which is explained next.

**Fully homomorphic encryption.** FHE [25] scheme supports any number of operations on any encrypted data. The circuit which is designed for FHE is homomorphically evaluated. FHE is suitable for any sort of application working with encrypted data. Due to computational overhead, FHE is less efficient than PHE and SWHE. As of today, for a particular application PHE and SWHE schemes are much more practical compared to FHE schemes.

In addition, based on the type of computations performed on the encrypted data, there are other categories which are discussed below [26].

**Additive Homomorphism** Additive homomorphic encryption systems support addition over encrypted data. Let the integers  $a$  and  $b$  be encrypted to  $E(a)$  and  $E(b)$ , respectively. Then, we can perform the sum of  $a$  and  $b$  homomorphically as follows

$$E(a) \circ (b) = E(a + b), \quad (2.1)$$

where  $\circ$  stands for the homomorphic operation over the ciphertexts, which results in the encrypted sum of those integers.

**Multiplicative Homomorphism** Multiplicative homomorphic encryption systems supports multiplication over encrypted data. Let two integers  $a$  and  $b$  be encrypted to  $E(a)$  and  $E(b)$ , respectively. Then, we can perform the product of  $a$  and  $b$  homomorphically as follows

$$E(a) * E(b) = E(a \times b), \quad (2.2)$$

where  $*$  stands for the homomorphic operation over the ciphertexts, which results in the encrypted product of those integers.

## 2.3 Lattice-based Cryptography

Lattice is a set of points in  $n$ -dimensional real space with a periodic structure. More formally, given  $n$ -linearly independent vectors  $(v_1, \dots, v_n) \in R^n$ , the lattice generated by them is the set of vectors with

$$L(v_1, \dots, v_n) := \left\{ \sum_{i=1}^n \alpha_i v_i \mid \alpha_i \in \mathbb{Z} \right\}. \quad (2.3)$$

The vectors  $v_1, \dots, v_n$  are known as a basis of the lattice. Lattice-based cryptography is the generic term for cryptographic primitives whose constructions involve lattices, either in the construction itself or in the security proof [27]. The first lattice-based public-key encryption scheme was introduced by Oded Regev in 2005 [28], whose security was proven under worst-case hardness assumptions. The computations involved in lattice-based cryptography are very simple and often require only modular and polynomial arithmetic. Recently lattice-based cryptography becomes practical even for resource-constraint computing platforms [29] although their relatively high key and ciphertext sizes are still an implementation concern.

## 2.4 Machine Learning Algorithms

Machine learning (ML) is the field of scientific study of algorithms and statistical models that computer system use to perform a specific task without using explicit instructions, relying on patterns and inference instead. ML provides systems with the ability to automatically learn and improve from experience without being programmed. The learning process begins with data or observations, such as examples, in order to search patterns in data and make better decisions in the future based on the past examples [30]. ML algorithms are often categorized as supervised or unsupervised.

**Supervised ML algorithms.** The majority of ML learning techniques uses supervised learning. In supervised learning a mathematical model of a set of data is built, where data contains both the inputs and the desired outputs [31]. A data is known as training data, and consist of a set of training examples, where each example is represented by a vector, called a feature vector. Supervised learning algorithms learn a function that can be used to make a decision for the output associated with new inputs, through iterative optimization of a loss function. Supervised learning algorithms include classification and regression. Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs might have any numerical value within a range.

**Unsupervised ML algorithms.** Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data

points. Therefore, the test data that the algorithm learns from, has not been labeled, classified or categorized. The algorithm specifies commonalities in the data and assigns a new piece of data to a cluster according to the presence or absence of such commonalities. Clustering is the assignment of a set of observations into clusters such that observations within the same cluster are similar, while observations drawn from different clusters are dissimilar [32].

Applying machine learning involves creating a model, which is trained on some training data and then can process additional new data to make predictions. Various types of models have been used and studied for machine learning system. We explain three such models, employed in this dissertation.

### 2.4.1 Decision Tree Algorithm

A decision tree is one of the most popular methods of classification as it is easy to be interpreted by humans. Decision tree classifiers (DTC's) are used in many diverse areas such as character recognition, medical diagnosis, and speech recognition, to name only a few. DTC is capable of breaking down a complex decision-making process into a collection of simpler decisions, which are easier to implement and interpret [33]. A decision tree is structure, in which an internal node represents a condition on an attribute, each branch denotes the outcome of the condition, and each leaf node stands for a class label (a decision taken after computing all attributes). A path from root to leaf are referred as a classification rule.

The decision tree can be linearized into decision rules [34], where the conditions along the path form a conjunction in an *if* clause, and the outcome (i.e., class label) is the content of the leaf node. In general, the rules have the following form:

$$\text{if } p_1 \wedge p_2 \wedge \dots \text{ then } label,$$

where the predicate  $p_i$  represents a condition and  $\wedge$  stands for the logical-AND operation.

### 2.4.2 Naïve Bayesian Algorithm

Naïve Bayesian classifier [35] is a probabilistic classifier based on the Bayes theorem [36], which is comparable in performance with those based on decision tree and neural networks depending on application and data set. Bayesian classifiers have high accuracy and speed when applied to large data sets. Naïve Bayesian classifier is a simple Bayesian classifier based on an assumption that the effect of an attribute value on a given class is independent of the values of other attributes. This assumption is called class conditional independence.

Naïve Bayesian classifier calculates the probability that a tuple (i.e., record in the context

of IDS) belongs to each class (suppose there are  $m$  classes) using posterior probability, where each tuple is represented by an  $n$ -dimensional attribute vector. The classifier will predict that the tuple belongs to the class having the highest posterior probability conditioned on the tuple.

### 2.4.3 Neural Network Algorithm

Neural network classification is based on the back propagation algorithm [37]. Neural network is a set of connected input/output units (also called neurons), where each connection has a weight associated with it. Actually, the knowledge is encoded in a set of numerical weights and biases. The network is able to predict the correct class label of an input vector by adjusting the weights during the learning phase. Neural networks have long training times, and their required parameters are typically best determined empirically. In addition, it is difficult to interpret numeric weights in term of rules, making it hard for the human to find out what the neural network has learned [38] by humans. Neural networks are able to classify patterns, on which they have not been trained. They can be used when there exists small amount of knowledge pertaining to relationship between attributes and classes. The computation process of neural network is amenable to parallelization techniques as algorithms used in neural network computations are inherently parallel.

The back propagation algorithm performs learning on a multilayer feed-forward neural network. A multilayer feed-forward neural network consists of one input layer, one or more hidden layers, and one output layer. The inputs to the network correspond to the attributes measured for each training tuple. These inputs pass through the input layer and then are fed simultaneously to a second layer, which is also known as hidden layer. In practice usually only one hidden layer is used. The weighted output of the hidden layer is input to the output layer.

## 2.5 Evaluation of Classification Algorithms

To evaluate the performance of any classification algorithm, we measure four values: i) **true positive (TP)** is an outcome where the model correctly predicts the positive class; ii) **false positive (FP)** is an outcome where the model incorrectly predicts the positive class; iii) **false negative (FN)** is an outcome where the model incorrectly predicts the negative class.; iv) **true negative (TN)** is an outcome where the model correctly predicts the negative class. In the context of NIDS, the positive class is an attack class while the negative class is the class pertaining to normal network traffic.

### 2.5.1 Evaluation Metrics

Based on these measurements, the performance of any classification algorithm is evaluated using four metrics:

1. **Accuracy Rate (AR):** The ratio of the number of correctly classified records to the number of all records

$$AR = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.4)$$

2. **Precision:** The ratio of the number of records correctly classified as attack to the total number of alarms generated by IDS

$$Precision = \frac{TP}{TP + FP} \quad (2.5)$$

3. **Recall (detection rate):** The ratio of the number of records correctly classified as attack to the total number of attacks

$$Recall = \frac{TP}{TP + FN} \quad (2.6)$$

4. **False Alarm Rate (FAR):** The ratio of the number of false alarms to the number of correctly classified records

$$FAR = \frac{FP}{TN + FP} \quad (2.7)$$

### 2.5.2 Precision-Recall Curves

Success of prediction can be measured using the precision-recall metric when the classes are very imbalanced. In information retrieval, precision can be defined as a measure of result relevancy, while recall is defined as the fraction of relevant results that are actually returned.

The precision-recall curve shows the trade-off between precision and recall for different thresholds, which are used for classification. A high large area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall). In a system, where recall rate is high but precision is low, many results are returned, but most of the predicted labels are incorrect. In a system with high precision but low recall, very few results are returned, but most of the predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly.

### 2.5.3 Confusion matrix

In the field of machine learning, a confusion matrix, also known as an error matrix [39], is a specific table as shown in Figure 2.2 for visualization of the performance of a classification algorithm, typically a supervised learning one. Each row of the matrix represents the number of instances in a actual class while each column represents the number of instances in a predicated class [40]. The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

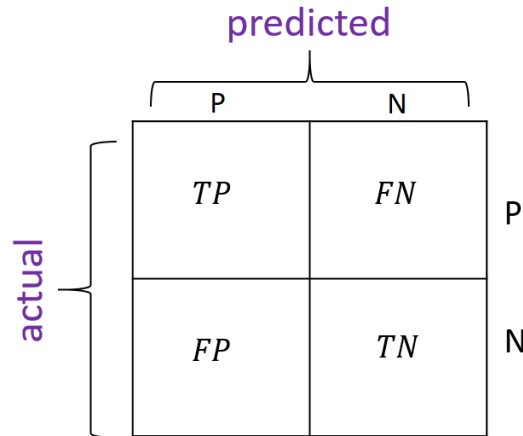


Figure 2.2: Confusion matrix

## 2.6 Data Pre-processing Techniques

The performance of classification algorithms depends on the data quality to a large extent. In particular, feature selection plays an exceedingly important role, which is explained in this section.

### 2.6.1 Attribute selection

Whether we select and gather sample data ourselves or it is provided to us by domain experts, the selection of attributes is critical in obtaining a correct model of the data under scrutiny. Including redundant attributes can be misleading to modeling algorithms. In addition, keeping irrelevant attributes in your data set can result in overfitting. For example, decision tree algorithms seek to make optimal splits in attribute values. For example, those attributes that are more correlated with the prediction are split on first (e.g, meaning that the more relevant attributes are placed higher up in the decision tree; the most relevant being at the root of the tree and splitting on first). That less relevant and irrelevant

attributes deeper in the tree are used to make prediction decisions may only be beneficial by chance in the training data set. This overfitting of the training data can negatively affect the modeling power of the method and deteriorate the predictive accuracy. Therefore, before evaluating machine learning algorithms it is important to remove redundant and irrelevant attributes from data set. One of the methods to remove those irrelevant attributes from a data set is known as *feature selection*; applied by navigating through all possible combinations of attributes and locating the best or a good enough combination that improves performance over selecting all attributes. The definition of “best” depends on the target problem, but typically means the one giving the highest accuracy.

Three are three key benefits for performing feature selection on data:

- Reduces Overfitting: Less redundant data reduces the probability of making decisions based on noise.
- Improves Accuracy: Less misleading data improves modeling accuracy.
- Reduces Training Time: Training algorithms run faster if they process less data.

Many feature selection techniques [41], such as those based on correlation, information gain, and learner are supported in WEKA, which is widely used open source software in data mining and machine learning applications [42]. We will briefly explain about each technique and illustrate some results we obtain by applying these techniques on ISCX 2012 data set in subsequent sections.

**Correlation-based Feature Selection** is more formally referred to as Pearson’s correlation coefficient in statistics [43,44], which is used for selecting the most relevant attributes in a data set. Correlations between each attribute and the output variable (i.e., class label) are calculated such that only those attributes that have a moderate-to-high positive or negative correlation (close to -1 or 1) are remained, and attributes with low correlations (value close to zero) are dropped.

**Information-Gain-based Feature Selection** is another popular feature selection technique [45]. Estimated information gain value (also called entropy) for each attribute and the output variable varies from 0 (no information) to 1 (maximum information). Thus, attributes with higher information gain are mostly preferred.

**Learner-based Feature Selection** is a popular feature selection technique that uses a generic but powerful learning algorithm and evaluates the performance of the algorithm on the data set where each time different subsets of attributes are selected. The features in the subset that results in the best performance are selected as the most related features. The algorithm used to evaluate the subsets does not have to be the algorithm that you intend to use to model your problem, but it should be generally quick to train and powerful, like a decision tree method.



In our experiments we adapt feature ranking methods such as the feature selection based on information gain) due to their simplicity and the fact that good success rates have been reported for them in the literature [46]. In feature ranking methods, a ranking criterion is used to score the feature, and a threshold is used for removing those features below the threshold. Actually, the relevance of the features are identified using these methods. A relevant feature is the one that might be independent of the input data, but cannot be independent of the class labels.

# Chapter 3

## RELATED WORK

Intrusion detection system (IDS), which has been the topic of a number of surveys and articles, can be classified in two types based on data source. Host intrusion detection systems (HIDS) collect data from a host computer and monitors activities such as file changes, system logs, and network traffic pertaining to the host [16]. Network intrusion detection systems (NIDS), which are also the focus of this dissertation, monitor packets in a network [15] and protect the system against malicious activities such as denial-of-service (DoS) attacks.

Both the specific machine learning algorithms and data sets employed in the training phase play important roles in the performance of any IDS. However, the quality and capacity of the training data set to capture the normal as well as anomalous behavior of a target system under protection are especially crucial in IDS performance.

### 3.1 Literature overview on data sets used in NIDS

Data sets play an important role in the training, testing and validation of any intrusion detection technique. The ability of a method to detect anomalous behavior is influenced by the quality of data to a large extent. The majority of research efforts in the area of NIDS is still based on the simulated data sets because of non-availability of real data sets. DARPA [4] and KDD Cup 1999 [47] are data sets that have been profitably utilized in the works of intrusion detection domain. However, their accuracy and ability to reflect real-world scenarios have been extensively criticized in such works as [48, 49] for the sole reason that they are outdated. One of the most important deficiencies of the KDD data set is the existence of large number of redundant records, which causes the learning algorithms to be biased towards frequent records. Thus, learning from infrequent, but influential records can be obstructed, which may play an deteriorating role in accuracy of the learned model for identifying attacks. In addition, the existence of these repeated records in the test set causes the evaluation results to be biased positively toward methods

which have better detection rates on the frequent records.

The evaluations of the existing data sets since 1998 show that most are out of date and unreliable to use. Some of these data sets suffer from the lack of traffic diversity and volumes, some do not cover the variety of known attacks, while others anonymize packet payload data, which cannot reflect the current trends. Some are also lacking feature set and metadata.

The literature review shows that three state-of-the-art data sets are currently used in network intrusion detection systems: Kyoto 2006+, ISCX IDS 2012, and CICIDS 2017. In what follows, we briefly explain each of these data sets.

**Kyoto 2006+** data set is built on three years of real network traffic data (between Nov. 2006 - Aug. 2009) which are obtained from diverse types of honeypots [50]. Kyoto 2006+ data set greatly contribute to the efforts of IDS researchers in obtaining more practical, useful and accurate evaluation results. In the data set 14 significant and essential features are extracted from the honeypot data based on the 41 original features of KDD Cup 99 data set. Furthermore, redundant and insignificant features are eliminated. Also, 10 additional features are included to enable investigating more effectively what kinds of attacks exist on the monitored networks. These features also can be utilized for NIDS evaluation. This data set has limited view of the network traffic as only the attacks which are directed at honeypots can be observed, which is an important drawback.

**ISCX IDS 2012** data set [5] is generated by Information Security Centre of Excellence at university of New Brunswick. It consists of labeled network traces, including full packet payloads in pcap format. It includes network traffic for HTTP, SMTP, SSH, IMAP, POP3, and FTP protocols with full packet payload. It consists of 7 days of network activity (normal and malicious). The labeled data is provided in XML format, whereby 20 features are available. ISCX data set covers common attacks such as DoS, DDoS, Brute Force, Port scan and Botnet. The problem with this data set is that it does not represent new network protocols since nearly 70% of current network traffic are HTTPS and there are no HTTPS traces in this data set. Also, simulated attack distribution is not based on real world statistics.

**CIC IDS 2017** data set [51] is also generated by Information Security Centre of Excellence at university of New Brunswick. It contains benign and the most up-to-date common attacks and resembles the true real-world data (PCAPs). They built the abstract behavior of 25 users based on the HTTP, HTTPS, FTP, SSH, and e-mail protocols. The data capturing period started at 9:00am, Monday, July 3, 2017 and ended at 17:00 on Friday July 7, 2017, for a total of 5 days. Monday is the normal day and only includes the benign traffic. The implemented attacks include Brute Force FTP, Brute Force SSH, DoS, Heart-bleed, Web Attack, Infiltration, Botnet and DDoS. They are executed both morning and afternoon on Tuesday, Wednesday, Thursday and Friday. The labeled data set is provided in CSV format file that has six columns as label for each flow namely FlowID, SourceIP,

DestinationIP, SourcePort, DestinationPort, and Protocol with more than 80 network traffic features.

In this dissertation, we take advantage of the last two data sets, ISCX IDS 2012 and CIC-IDS 2017 data set [51]. The latter data set is also publicly available (<http://www.unb.ca/cic/data/sets/IDS2017.html>).

## 3.2 Literature overview on rule-based machine learning

In the literature, various machine learning techniques are frequently used in IDS solutions to form detection rules [52]. More specifically, the goal of machine learning is to generate a minimal rule set, which is learned from historical data and can distinguish signatures of various attacks. To this end, a number of machine learning techniques such as  $k$ -nearest neighbor [14], decision trees [33], the Naïve Bayes method [53], artificial neural networks [54], and support vector machines [55] have been profitably utilized. This dissertation is mainly focused on decision trees, naïve Bayes, and artificial neural networks to generate rules for intrusion detection applications.

There are a couple of works on rule-based naïve Bayesian classification techniques. In [56] the authors provide a simple three-step methodology for constructing a rule set using Naïve Bayesian classification. In their approach the data set is scanned only once, at the time of building the rule set. To classify a new record, the set of classification rules is scanned, and the rule which is satisfied by the record is said to be fired to determine the class of the record. Thus, subsequent scanning of the data set for each new record is avoided<sup>1</sup>. Also, in [57] the authors propose an algorithm that convert naïve Bayes models with multi-valued attribute domains into sets of rules, which have the form **IF**  $r_i = (a_1^i, \dots, a_d^i)$  **THEN**  $x_j$ , where  $(a_1^i, \dots, a_d^i)$  and  $x_j$  are attribute and class values, respectively. They use labeling to represent strength of each rule. They formalize this by defining a label and a function transforming conditional probabilities into labels. They use a pruning method provided in [58] to eliminate rules with low significance.

Also, there are some studies on rule-based neural network classification. In [59–61], they utilize an information-theoretic algorithm for constructing rules from the training data. Then, the rules are used to construct a neural network to perform posterior probability estimation. The advantage of this method is that new data can be incorporated without repeating the training on previous data. In [61], they propose a network architecture, which acts as parallel Bayesian classifier, but can also compute posterior probabilities of the class variables. They take advantage of information theory to learn only the most important rules. The proposed architecture avoids repetitive network training processes by specifying weights in terms of probability estimates derived from the training data.

---

<sup>1</sup>When all the rules are extracted, for a new record there is no need to scan the whole data set to determine its class, only the extracted rules are examined to find to which rule(s) the new record is matched.

In addition, the number of rules which are learned from the data, automatically specify the number of hidden nodes of the network; consequently, there is no need to specify the number of such nodes in advance. In this dissertation we benefit from [56] for naïve bayes and [61] for neural networks to implement rule-based classification for intrusion detection system.

### 3.3 Literature overview on similar works

Privacy issues related to the application of IDS to system data have not been studied extensively in the literature. The works [62, 63] propose using pseudonyms, which are generated in cooperation with a trusted third party (TTP) or host computer under monitoring itself. They substitute pseudonyms for any user identifying information within the collected data.

In [2] the authors propose a pseudonym-based privacy preserving method for intrusion detection system, named PPIDS, by applying cryptographic methods to log files without a trusted third party. Using cryptographic methods, PPIDS can prevent users' log information from being monitored and misused. In addition, PPIDS can provide anonymity (encryption of ID), pseudonymity (encryption of quasi-identifiers such as IP address), confidentiality of data, and unobservability. However, PPIDS cannot provide perfect unlinkability as a deterministic algorithm is used for encryption and therefore one can still infer behavioral patterns pertaining to a specific user. Their scenario is different from ours: they assume that one party knows both data and intrusion policies, but due to its low computational power, detection operation is performed by another party.

In this dissertation, we focus on the problem of private evaluation of intrusion detection models, where it is assumed that one of the parties holds a trained attack model while the other holds data to it. There is a paucity of works in the literature addressing this specific subject. The work in [64] describes a fairly generic protocol for private evaluation of decision trees, performed in two phases; namely a comparison phase followed by an evaluation phase. The tree is viewed as a polynomial in the decision variable, which is evaluated using a SWHE scheme [8, 65]. The client encrypts its data and sends the ciphertext to the server that performs the private evaluation operation. The server and the client run an interactive protocol for comparison operation for every node in the decision tree. While the proposed solution is effective to classify an input, it may not be highly efficient for large decision trees as the comparison protocol must be run many times, which increases computation and communication overhead, thus the total classification time.

In another work [66], the authors propose a protocol for private evaluation of decision trees, whereby it is assumed that one of the parties holds a trained decision tree. The protocol is based on additive homomorphic encryption and oblivious transfer protocol and it

is secure against semi-honest adversaries. They also modify the protocol to provide security against malicious adversaries. They perform the decision tree evaluation protocol on five real data sets from the UCI repository [67]. In the semi-honest case for the “housing” data set, their protocol can evaluate a 13 dimensional feature vector on a tree with 92 decision nodes in around 4 minutes and 1.8 MB of communication. On a tree with 47 decision nodes and 20 dimensional feature vector, our protocol completes in 30 seconds and require about 128 KB of communication<sup>2</sup>. Furthermore, for the “breast-cancer” data set, the protocol in [66] can evaluate a 9 dimensional feature vector on a tree with 12 decision nodes in around 0.54 seconds and 205 KB of communication. On a similarly sized tree over an equally large feature vector, our protocol completes in 0.14 seconds and requires about 128 KB of communication, representing  $4\times$  and  $1.6\times$  improvements in computation and bandwidth, respectively.

In case of handling malicious client, on the “breast-cancer” data set, the protocol in [66] for a single input completes in 12.3 s and requires 8.2 MB for communication. For the “housing” data set, their protocol completes in 357 s and requires 256 MB of communication. We explain how our protocol deals with malicious client (DO) as well as malicious server (SOC) in Chapter 6.

Furthermore, neither in the protocol in [66] nor in our protocol based on decision tree, the client, who owns input, learns the features used in the decision nodes.

Generally speaking our protocol based on decision tree performs private evaluation of decision trees for intrusion detection. It utilizes lattice-based cryptography that allows somewhat fully homomorphic operations over the encrypted data. A Simple Encrypted Arithmetic Library-SEAL v2.2 [68] (SEAL) is used in this dissertation as the state-of-the-art homomorphic encryption solution. SEAL has been publicly released and can be downloaded for experimentation and research purposes<sup>3</sup>.

---

<sup>2</sup>We do not include the time spent on encrypting the decision tree and exclude the bandwidth used to send them as encryption and transmission of the tree are performed once in our setting. Also, our execution time includes only the homomorphic evaluation of the decision tree excluding other less costly operations such as the decryption of the results.

<sup>3</sup>SEAL is freely available at <http://sealcrypto.org>

# Chapter 4

## Rule-based Classification Techniques

The rule-based classification techniques utilizes various methods such as probabilistic and information-theoretic algorithms for generating rules from the training data. These rules are then used to detect malicious behaviors in the testing data set. Broadly speaking, for building rule-based classifiers, there are two generic approaches as follows:

1. **Direct approach** In this approach, rules are extracted directly from the training data set. Sequential covering algorithms [69] constitute prominent example of common direct methods for building classification rules. In sequential covering algorithms, rules are extracted sequentially, i.e., one at a time, starting with an empty set of rules. Each time a rule is extracted, all records in the training set covered by the rule are removed. Example algorithms include CN2 and RIPPER [70], which are commonly used direct methods for building classification rules.
2. **Indirect Approach** In this approach, rules are extracted using classification techniques (e.g., decision trees, naïve Bayes algorithm, neural networks, etc.).

We consider a problem of constructing a classifier by relating a set of  $d$  discrete feature variables to a discrete class variable  $X$ . To be precise, let the set  $\mathcal{A} = \{a_1, \dots, a_d\}$  be the  $d$  discrete feature variables, and each variable  $a_i$  can take discrete values from the alphabet  $\mathcal{V}_i = \{v_{i,1}, \dots, v_{i,t_i}\}$ , where the cardinality of  $\mathcal{V}_i$  is  $t_i$  for  $1 \leq i \leq d$ . For simplicity let  $v_{i,k}$  represent a specific value in  $\mathcal{V}_i$ , i.e.,  $v_{i,k} \in \mathcal{V}_i$  and it may be the case that  $a_i = v_{i,k}$ . We define  $X$  as the class variable with a discrete alphabet  $\mathcal{X} = \{x_1, \dots, x_m\}$ , where  $m$  is the number of classes. A training set consists of  $n$  data vectors of the form  $r_j = \{a_1(j), \dots, a_d(j), x(j)\}$ ,  $1 \leq j \leq n$ , where  $a_i(j) \in \mathcal{V}_i$  and  $x(j) \in \mathcal{X}$  for the  $j$ -th record in the data set.

Let the vector  $(v_{1,k_1}, \dots, v_{d,k_d})$  represent a typical record for feature vector  $\{a_1, \dots, a_d\}$ . A rule, then, can be defined by some arbitrary joint conjunction function  $F(a_1, \dots, a_d)$ ,  $\ell \leq d$  for a particular class  $x_j$ ,  $1 \leq j \leq m$ , in the form of

$$\mathbf{IF} (a_1 = v_{1,k_1} \wedge \dots \wedge a_\ell = v_{\ell,k_\ell}) \mathbf{THEN} X = x_j.$$

A data vector is said to satisfy a rule if the conjunction in the left-hand-side (LHS) of the rule in the vector holds. The number of features on the LHS of the rule ( $l$ ) is called the rule order. In this dissertation we adapt three classification techniques; namely binary decision tree, naïve Bayes, and neural networks to extract intrusion rules from the training data set. We compare the implementation results of the three classification algorithms in Chapter 7.

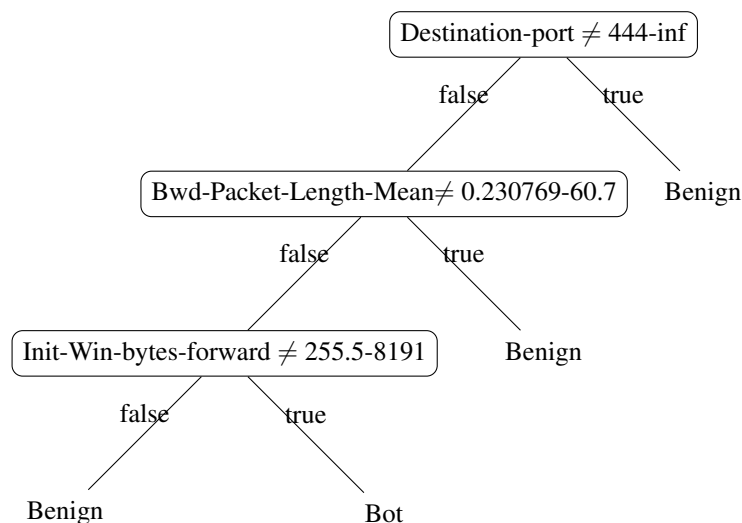
## 4.1 Decision Tree-based Model

Decision tree classifiers form a popular category of classification; they are easy to understand and known for their accuracy in many applications. Although the size of decision tree can become large causing difficulties in its interpretation at times, the rules based on IF-THEN are still accessible to human understanding. In this subsection, we look at how to build a rule-based classifier by extracting IF-THEN rules from a decision tree.

In a decision tree, one rule is generated for each path from the root to a leaf node. Along a given path, each splitting criterion is logically ANDed to form the left-hand-side (LHS)/“IF” part of the rule. The right-hand-side (RHS)/“THEN” part of the rule is the class prediction held in the leaf node [36]. One of the advantages of decision trees is that no rule conflicts is possible as no two rules will be triggered for the same record. We have one rule for every leaf, and any record can map to only one leaf.

Figure 4.1 demonstrates the binary decision tree model for a sample from the CIC-Friday-Morning data set, where there are 7 features excluding the class attribute. The number of records in the training and test sets are 429 and 186, respectively; and there are two classes: “Benign” and “Bot”.

Figure 4.1: Binary decision tree as a detection model.



The decision tree of the Figure 4.1 can be converted to IF-THEN rules by tracing the path



from the root node to each leaf node in the tree. The extracted rules are as follows:

$R_1$  : **IF** (Destination-Port  $\neq$  444-inf) **THEN**  $X = \text{Benign}$

$R_2$  : **IF** (Destination-Port = 444-inf)  $\wedge$  (Bwd-Packet-Length-Mean  $\neq$  0.230769 – 60.7)  
**THEN**  $X = \text{Benign}$

$R_3$  : **IF** (Destination Port = 444-inf)  $\wedge$  (Bwd Packet Length Mean = 0.230769-60.7)  $\wedge$   
(Init\_Win\_bytes\_forward = 255.5-8191) **THEN**  $X = \text{Benign}$

$R_4$  : **IF** (Destination Port = 444-inf)  $\wedge$  (Bwd Packet Length Mean = 0.230769-60.7)  $\wedge$   
(Init\_Win\_bytes\_forward  $\neq$  255.5-8191) **THEN**  $X = \text{Bot}$

Here, the rule  $R_4$  can be used to detect the attack type “Bot”. The complexity of rules depends on the depth of the tree and number of leaf nodes.

## 4.2 Naïve Bayesian-based Model

Another popular classification technique is naïve Bayesian [36], which is a probabilistic classification approach using the Bayesian theorem to predict the classes of unclassified records. In this approach, we will construct a set of classification rules on top of the naïve Bayesian classifier. We will demonstrate by experiments that this technique works as expected to extract intrusion rules from the network training data set. Whenever a new record is to be classified, the set of classification rules is searched to explore the rule that is satisfied by the record. One of the advantages of rule-based naïve Bayesian classifier in comparison with naïve Bayesian classifier is that any time a new data record is to be classified, the entire data set does not need to be scanned, which is normally a very costly step if the data set is very large [56]. Also, rule-based naïve Bayesian classification usually yields a high degree of classification accuracy for many applications.

In this section we show how naïve Bayesian classification is used for building a rule-based classifier. Naïve Bayesian classification is based on the Bayesian theorem, whereby the probability that a new record  $r$  belongs to the class  $x_j$  can be estimated using Equation 4.1.

$$P(x_j|r) = \frac{P(r|x_j)P(x_j)}{P(r)} \quad (4.1)$$

Here,  $P()$  denotes the probability whereas  $P(x_j|r)$  stands for the conditional probability of  $x_j$  given that  $r$  has occurred, and  $x_j$  is in the set of classes  $\mathcal{X} = \{x_1, x_2, \dots, x_m\}$  that is used to classify the data. For a new unclassified record  $r$ , Equation 4.1 is computed for every class  $x_j$ . The class, whose conditional probability  $P(x_j|r)$  is highest, is selected as the prediction for the record’s class. Since the denominator  $P(r)$  is constant across all classes, it can be removed from the computations. Consequently, the simplified formula

in Equation 4.2 can be used to predict the class with the highest probability.

$$P(x_j|r) \sim P(r|x_j)P(x_j), \quad (4.2)$$

Where the symbol  $\sim$  is used to indicate that the LHS is proportional to the RHS. In addition, the feature values in the record are assumed to be independent of each other following the naïve Bayesian classification assumption; thus, if  $r$  is a record of  $d$  independent events (i.e., feature values in our case)  $\langle v_{1,k_1} \wedge v_{2,k_2} \wedge \dots \wedge v_{k,k_d} \rangle$ , then  $P(r|x_j)$  in Equation 4.2 can be approximated as shown in Equation 4.3.

$$P(r|x_j) = \prod_{i=1}^d P(v_{i,k_i}|x_j) \quad (4.3)$$

Algorithm 1 describes the steps needed to extract classification rules based on naïve Bayesian classification. We assume that each feature  $a_i$  is limited to  $t_i$  number of distinct values. If the data set contains continuous values, a pre-processing step is needed for discretization of each feature to  $t_i$  distinct values, which is a popular pre-processing technique in data mining.

---

**Algorithm 1** Generation of rules based on naïve Bayesian classification

---

**Input:**  $\mathcal{S} = \{s_1, s_2, \dots, s_{n_c}\}$ , where  $s_i$  are instances

**Output:**  $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_u\}$ , where  $\mathcal{R}_z$  is set of rules, whose RHS results in  $x_z$  for  $1 \leq z \leq u$ .

```

1: for  $s_i \in \mathcal{S}$  do
2:   for  $x_j \in \mathcal{X}$  do
3:      $p_{i,j} \leftarrow P(x_j|s_i)$ 
4:   end for
5:    $p_{i,z} \leftarrow \max(p_{i,1}, \dots, p_{i,u})$ 
6:    $\mathcal{R}_z \leftarrow \mathcal{R}_z \cup s_i$ 
7: end for
8: return  $\mathcal{R}$ 

```

---

Before Algorithm 1 is executed, its input  $\mathcal{S}$ , the set of all possible combinations of feature values, is generated. In order to find  $\mathcal{S}$ , we have to identify the set of all distinct values for each feature (i.e. the domain of the feature). The number of all possible combinations, i.e., the cardinality of  $\mathcal{S}$ , can be found as shown in Equation 4.4.

$$n_c = \prod_{i=1}^d t_i \quad (4.4)$$

These  $n_c$  combinations (from now we call instances) are the inputs for the Algorithm 1.

For example,  $s_1 = (v_{1,1}, v_{2,1}, \dots, v_{d,1})$  while  $s_{n_c} = (v_{1,t_1}, v_{2,t_2}, \dots, v_{d,t_d})$ .

In Steps 2 through 4 of the algorithm, the probability that the instance  $s_i$  is classified as the class  $x_j$  is calculated for all classes using Equation 4.2. At Step 5, the maximum probability for the instance  $s_i$ ,  $p_{i,z}$  is determined and the instance is added to the set  $\mathcal{R}_z$  in Step 6, which includes the instances that belong to the class  $x_z$ . The **IF** part of the rule checks whether the new record is one of the instances in  $\mathcal{R}_z$ ; if so the **THEN** part predicts the record in the corresponding class. Following our former notation,  $\mathcal{R}_z$   $1 \leq z \leq u$  is the set of rules for the class  $x_z$ .

Using the rule-based technique, whenever a new record  $r$  is to be examined for class prediction, there is no need to scan the existing data set to apply the Bayesian technique to figure out the class. Instead, the rule whose conditions are met by the data in the new record is fired to infer the class. This is done by checking whether  $r \in \mathcal{R}_z$  for  $1 \leq z \leq u$ , which yields only one class.

Whenever several new records are added to the data set, the existing classification rules might not reflect precisely what exists in the data set. Thus, the rule set should be refreshed periodically, where the frequency can be set by the administrator of the system depending on the application requirements. The refreshing process will not degrade performance since it can be performed off-line. In our context, SOC does not learn from the data provided by DO as it does not really see the content of DO's data. Rather, it uses its own resources to find new records and updates its rule set periodically.

In the following we show some sample rules, which are extracted from the CIC-Friday-Morning sample data set using the naïve Bayesian approach. The features of the data set and the domain of each feature is illustrated in Table 4.1. There are seven features, where each feature can take five distinct values.

The total number of combinations according to the domain of the features can be calculated by multiplying the number of values in each domain, i.e.,  $5^7 = 78125$  since each feature takes five discrete values. Table 4.2 shows the class probability value for some of the combinations derived from CIC-Friday-Morning sample data set. Since the number of combinations is high, only a few combinations are illustrated. The training data set of CIC-Friday-Morning samples, which contains 429 records, is used as a basis to compute  $P(\text{Benign})$  and  $P(\text{Bot})$ .

Classification rules can be generated from the instances in Table 4.2, where the **IF** part represents the conditions on all values and the **THEN** part of the rule designates the class with the highest probability. As an example, the class probabilities for the first instance are  $P(\text{Benign}) = 0.0094$  and  $P(\text{Bot}) = 0.991$ . Since  $P(\text{Bot})$  is greater than  $P(\text{Benign})$ , the first instance can be considered as an intrusion rule for the Bot class, which can be formulated as follows:

Feature $a_i$	Domain $v_{i,k}$
Destination Port	-inf-37.5 37.5-66.5 66.5-416 416-444 444-inf
Bwd Packet Length Mean	-inf-0.230769 0.230769-60.7 60.7-110.125 110.125-197.875 197.875-inf
Avg Bwd Segment Size	-inf-0.230769 0.230769-60.7 60.7-110.125 110.125-197.875 197.875-inf
min_seg_size_forward	-inf-10 10-24 24-30 30-36 36-inf
Init_Win_bytes_forward	-inf-0.5 -0.5-255.5 255.5-8191 ,8191-16320 16320-inf
Subflow Bwd Packets	-inf-0.5 0.5-1.5 1.5-2.5 2.5-10.5 10.5-inf
Total Backward Packets	-inf-0.5 0.5-1.5 1.5-2.5 2.5-10.5 10.5-inf

Table 4.1: CIC-Friday-Morning sample data set's features and their domain.

**IF**(DestPort=444-inf)  $\wedge$  (wd\_Packet\_Length\_Mean=0.230769-60.7)  
 $\wedge$  (Avg\_Bwd\_Segment\_Size=0.230769-60.7)  $\wedge$  (min\_seg\_size\_forward=10-24)  
 $\wedge$  (Init\_Win\_bytes\_forward=-0.5-255.5)  $\wedge$  (Subflow Bwd Packets=0.5-1.5)  
 $\wedge$  (Total Backward Packets= 0.5-1.5) **THEN** Bot.

The rest of the classification rules are derived similarly. Since, our aim is to find the rules representing the Bot class, we only consider those rules, whose RHS designates the class

Table 4.2: Class probability values for some of the combinations derived from CIC-Friday-Morning

DestPort	Bwd_Packet_Length_Mean	Avg_Bwd_Segment_Size	min_seg_size_forward	Init_Win_bytes_forward	Subflow Bwd Packets	Total Backward Packets	P(Benign)	P(Bot)
444-inf	0.230769-60.7	0.230769-60.7	10-24	-0.5-255.5	0.5-1.5	0.5-1.5	0.0094	0.991
444-inf	197.875-inf	197.875-inf	36-inf	16320-inf	1.5-2.5	1.5-2.5	0.999	0.0
66.5-416	197.875-inf	0.230769-60.7	30-36	8191-16320	2.5-10.5	1.5-2.5	1.0	0.0
444-inf	0.230769-60.7	0.230769-60.7	10-24	-0.5-255.5	10.5-inf	2.5-10.5	0.0048	0.995
444-inf	0.230769-60.7	0.230769-60.7	10-24	8191-16320	10.5-inf	10.5-inf	0.023	0.977
444-inf	0.230769-60.7	0.230769-60.7	10-24	8191-16320	10.5-inf	0.5-1.5	0.0098	0.990
444-inf	0.230769-60.7	0.230769-60.7	10-24	-0.5-255.5	10.5-inf	10.5-inf	0.051	0.95
444-inf	0.230769-60.7	0.230769-60.7	10-24	255.5-8191	0.5-1.5	0.5-1.5	0.0784	0.921
444-inf	0.230769-60.7	0.230769-60.7	10-24	255.5-8191	0.5-1.5	2.5-10.5	0.0178	0.982
444-inf	0.230769-60.7	0.230769-60.7	10-24	255.5-8191	0.5-1.5	0.5-1.5	0.0784	0.921
444-inf	0.230769-60.7	0.230769-60.7	10-24	-0.5-255.5	10.5-inf	10.5-inf	0.0509	0.949
444-inf	0.230769-60.7	0.230769-60.7	10-24	-0.5-255.5	10.5-inf	2.5-10.5	0.00479	0.995
444-inf	0.230769-60.7	0.230769-60.7	10-24	-0.5-255.5	10.5-inf	0.5-1.5	0.0220	0.978
444-inf	0.230769-60.7	0.230769-60.7	10-24	255.5-8191	10.5-inf	0.5-1.5	0.168	0.831
...	...	...	...	...	...	...	...	...

Notes: Each instance represent one possible combination of all attribute values in the data set.

Bot. To give an idea, the number of rules which are extracted for the Bot class in the training set of the CIC-Friday-Morning data set is 54.

## 4.3 Neural Network-based Model

In this part we first explain how to learn a set of sufficiently good rules from the training data. Then, we construct a neural network using these rules to refine the rules to posterior probability estimation.

### 4.3.1 Learning Rules

Here we discover a set of rules from training set using information-theoretic measure [60]. Suppose the rule  $R_i$  of the form

**IF**  $\rho_i$  **THEN**  $x_j$  with probability  $p_{i,j}$ ,

where  $p_{i,j}$  is the conditional probability  $p(x_j|\rho_i)$ ,  $\rho_i$  is the conjunction of feature values in LHS of the rule, and  $x_j$  is the value of the class variable in the RHS.

A rule is considered good if the joint conjunction of feature values in the LHS is highly correlated with the RHS. We take an information theoretic approach, and measure the goodness of such a rule using the average number of bits of information that the occurrence of LHS of the rule gives about its RHS. In the following, we introduce the  $J$ -measure, developed by Smyth and Goodman [71], based upon the work of Blachman [72], which is defined as in Equation 4.5.

$$J(X; \rho_i) = P(\rho_i) \left( P(x_j|\rho_i) \log \frac{P(x_j|\rho_i)}{P(x_j)} + P(\bar{x}_j|\rho_i) \log \frac{P(\bar{x}_j|\rho_i)}{P(\bar{x}_j)} \right) \quad (4.5)$$

Here  $P(x_j|\rho_i)$  is the conditional probability of the class  $x_j$  given the joint conjunction of features  $\rho_i$  whereas  $P(\bar{x}_j|\rho_i)$  stands for the total conditional probability of all other classes, except for  $x_j$ , given  $\rho_i$ .

The  $J$ -measure is the product of two terms. The first term is  $P(\rho_i)$ , the probability that the LHS of the rule occurs in the data set. In order for a rule to be considered useful, the left-hand side should occur frequently. The second term is the cross-entropy of  $X$  and  $X$  given  $\rho_i$ , which is the measure of the goodness of fit between a posterior belief about  $X$  and a priori belief. There is a trade-off between accuracy and generality, where the high-order rules (less probable) are accurate predictors; however low-order rules (more probable) are less accurate.

The probabilities in Equation 4.5 are calculated for the training data set. The most informative rules are those with the greatest  $J$ -measure.

Now, we need a search algorithm to find all possible rules, and then select the best rules

using the  $J$ -measure. Several search algorithms exist, including the search of all possible rules [73], whose size is exponential in the number of features. In this dissertation, we use the method, proposed in [60] and described in Algorithm 2, to search in a smaller set using the records directly from a training set as a rule template.

Algorithm 2 is applied for each rule template (each example from the training set) independently, where each rule is order of  $d$  (the number of features in the training set excluding the class attribute).

---

**Algorithm 2** Rule generation using  $J$ -measure

---

**Input:** Training data set  $\mathcal{D} = \{r_1, r_2, \dots, r_n\}$

**Output:** Rule set  $\mathcal{R}$

```

1: for  $r_i \in \mathcal{D}$  do
2:   while True do
3:     Take  $r_i$  as the rule template (i.e., IF  $a_1(i) \wedge \dots \wedge a_d(i)$  THEN  $x(i)$ )
4:     Calculate the  $J$ -measure for the rule. Call this rule the parent rule.
5:     Generate child rules removing one attribute from the parent rule (if the parent
       rule was order  $k$ , each of  $k$  child rule is order  $k - 1$ )
6:     Calculate the  $J$ -measure for each child rule
7:     Choose the rule among the parent rule and the set of child rules with the greatest
        $J$ -measure
8:     Apply special cases:
9:     If two rules have the same  $J$ -measure, choose the one with the lower order
10:    If two rules with the same order have the same  $J$ -measure, choose one at a
       random
11:    If the chosen rule is not the parent rule, the chosen rule becomes a new parent
       rule; repeat the process starting at Step 3. If the chosen rule is the parent rule,
       add it to  $\mathcal{R}$  and break
12:   end while
13: end for
14: Remove duplicate rules
15: Return  $\mathcal{R}$ 

```

---

The algorithm for the initial rule set, in which the rule order decreases, must terminate in  $\mathcal{O}(d)$  iterations, where  $d$  is the number of features excluding class attribute. To learn rules incrementally, we perform Algorithm 2 on each record in the training set.

Also, before generating a neural network, duplicate rules must be removed to reduce the complexity of the network generated. In such case, the number of occurrences of the rule should be added to the remaining copy of the rule. In this way we save the statistics of the original data set. Thus, if a rule has  $k$  duplicates, its occurrence will be  $k + 1$ .

### 4.3.2 Classification

In this section, we show how a neural network normally estimates the class probabilities, given a set of feature values. We have a rule set  $\mathcal{R}$ , which is obtained using Algorithm 2 adopted from [60]. These rules can be used to compute the posterior probability of each class. A subset of rules  $\mathcal{F} \subseteq \mathcal{R}$  are fired for an input vector  $\{a_1, \dots, a_d\}$ , if the feature values in the input vector are the same as the left-hand sides of those rules. We estimate the log posterior probability using the formula in [61] for each output class  $X = x_j$  for  $1 \leq j \leq m$  as

$$\log P(x_j | \rho_1, \dots, \rho_{|\mathcal{F}|}) = \mathcal{C} + \log P(x_j) + \sum_{i=1}^{|\mathcal{F}|} w_{ij}, \quad (4.6)$$

where  $w_{ij} = \frac{P(x_j | \rho_i)}{P(x_j)}$  is the weight of the rule  $R_i \in \mathcal{F}$  with  $\rho_i$  on the LHS and  $x_j$  on the RHS. In the absence of any rules fired for an input vector (i.e.,  $|\mathcal{F}| = 0$ ), each class estimation is computed by the bias value  $\log(P(x_j))$ , which is the prior probability of the class  $x_j$ . Once the posterior probability is calculated for each class, the classification decision can be made by choosing the class with the largest probability.

The sum of values which activate the third-layer node given by

$$\sigma_j = b_j + \sum_{i=1}^{|\mathcal{F}|} w_{ij} \quad (4.7)$$

is exponentiated to produce the node output (Equation 4.8), where  $b_j = \log P(x_j)$ .

$$O'_j = e^{\sigma_j} = e^{-\mathcal{C}} \times P(x_j | \rho_1, \dots, \rho_{|\mathcal{F}|}) \quad (4.8)$$

The output of the exponentiation is then fed into a normalization layer that is formulated as

$$O_j = \frac{O'_j}{\sum_{k=1}^m O'_k}, \quad (4.9)$$

for  $1 \leq j \leq m$ . This effectively removes the constant  $\mathcal{C}$  from each input  $O'_j = e^{\sigma_j} = e^{-\mathcal{C}} \times P(x_j | \rho_1, \dots, \rho_{|\mathcal{F}|})$  and yields the posterior probability  $O_j = P(x_j | \rho_1, \dots, \rho_{|\mathcal{F}|})$  as desired.

The architecture that we use takes a three-layer feed forward network as illustrated in Figure 4.2. The first layer nodes correspond to input feature values in the input feature space. The second layer (also known as hidden layer) consists of  $|R|$  conjunctive nodes, one for each rule from the set of rules  $R$ . The output layer consist of one node for each class, that essentially computes the sum in Equation 4.7. In other words, an output node sums weights  $w_{ij}$  of the rules that have fired for each class and its bias (i.e.,  $b_j = \log P(x_j)$ ). After normalization, each node outputs an estimation for the posterior probability for each



class.

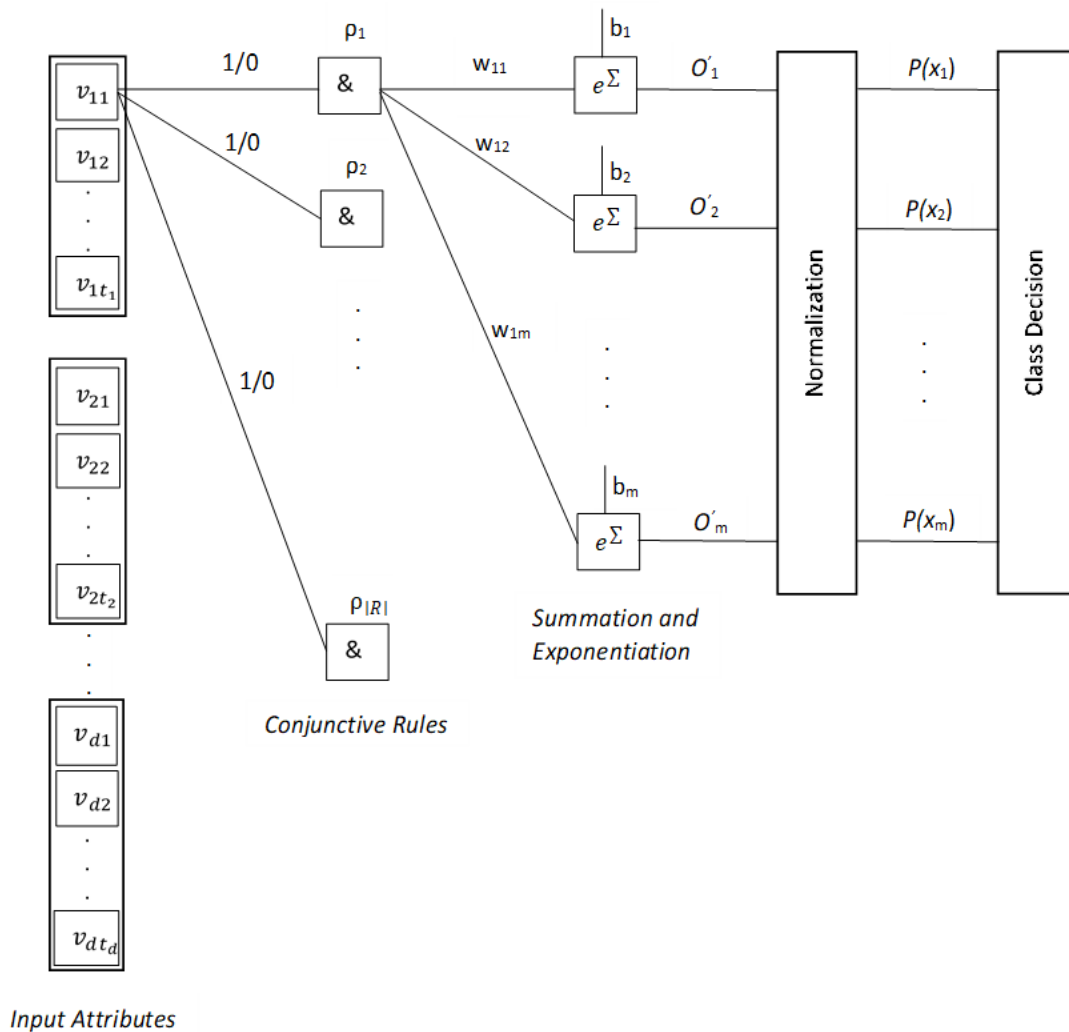


Figure 4.2: General architecture of rule-based neural network

Using the formula in Equation 4.6, a simple neural network with two classes can be constructed as in Figure 4.3. This neural network contains three layers, where the blue, grey, and red nodes represent the input, conjunctive rules, and output layers, respectively. Each blue node in the input layer represents one feature except for the class feature. Each grey node in the second layer represents a rule obtained from Algorithm 2, in which the nodes are connected to input nodes of the features in the LHS of the rule, which they represent; a second-layer-node is activated if the LHS of the rule is satisfied. The output layer contains one node for each value of the class feature. Each node (e.g., rule  $\rho_i$ ) from the second layer is connected to the third layer node  $x_j$  with a weight  $w_{ij}$ .

**Example 4.1.** Assume that for an input vector  $\{a_1, \dots, a_d\}$ , only the rules  $\rho_1, \rho_2$ , and  $\rho_3$  are fired. Assume also that there are two different classes. First, the probability for being classified as each of these classes must be calculated. Then, the class with higher probability will be chosen as class decision for the input vector. The probability estimation

for each class value is based on the number of rules fired for the class. The log posterior probability is measured for each class as follows:

$$\log P(x_1|\rho_1, \rho_2) = \mathcal{C} + \log P(x_1) + w_{11} + w_{21}$$

$$\log P(x_2|\rho_2, \rho_3) = \mathcal{C} + \log P(x_2) + w_{22} + w_{32}.$$

The class with higher posterior probability is selected as classification decision.

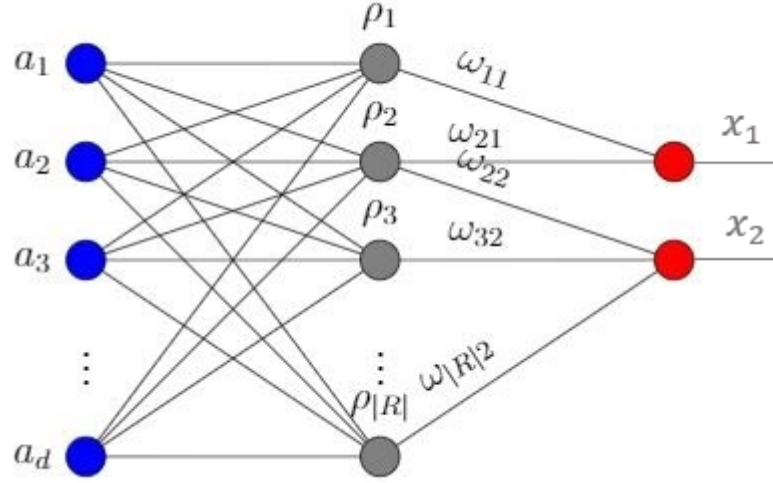


Figure 4.3: A simple architecture of the rule-based neural network.

### 4.3.3 Rule-based Intrusion Detection Using Rules from Neural Network

In order to utilize neural network technique in intrusion detection, we can use two different approaches. In the first approach, DO can homomorphically compute  $\sum_{i=1}^{|\mathcal{F}|} w_{ij}$  in Equation 4.7 using the encrypted rules and their weights, which are sent by SOC. The encrypted results for all classes are then sent to SOC that can compute the estimation for posterior probabilities of the classes. However, this will increase the communication overhead as well as the computation complexity.

In the second approach, we can use the neural network on the training data to refine the rules obtained in Algorithm 2. Indeed, Algorithm 2 can yield rules that can have the same LHS with different RHS. Also, one rule for a class can contain a subset of the LHS of another rule for a different class. Consequently, the rules from Algorithm 2 should be filtered. Thus, we select a subset of rules from  $\mathcal{R}$  for the attack class  $x_z$ . For the selection process we use Algorithm 3, that returns the attack rules  $R_z$  as well as the rules for normal records,  $R_{\bar{z}}$ .

---

**Algorithm 3** Generation of rules for the class  $x_z$  using posterior probability

---

**Input:**  $\mathcal{R} = \{R_1, \dots, R_i, \dots\}$ : is a rule set, where  $R_i = \mathbf{IF} \rho_i \mathbf{THEN} x_j$

**Output:**  $\mathcal{R}_z$ : rules which are designated to the attack class  $x_z$ .

```
1:  $\mathcal{R}_z \leftarrow \emptyset$ 
2:  $\mathcal{R}_{\bar{z}} \leftarrow \emptyset$ 
3: for  $R_i \in \mathcal{R}$  do
4:    $w_{ij} \leftarrow \frac{P(x_j|\rho_i)}{P(x_j)}$ 
5:   if  $(x_j = x_z)$  then
6:      $\mathcal{R}_z \leftarrow \mathcal{R}_z \cup R_i$ 
7:   else
8:      $\mathcal{R}_{\bar{z}} \leftarrow \mathcal{R}_{\bar{z}} \cup R_i$ 
9:   end if
10: end for
11: for  $R_i \in \mathcal{R}_z$  do
12:    $\bar{\mathcal{R}} \leftarrow \emptyset$ 
13:   for  $R_k \in \mathcal{R}_{\bar{z}}$  do
14:     if  $(\rho_k \subseteq \rho_i)$  then
15:        $\bar{\mathcal{R}} \leftarrow \bar{\mathcal{R}} \cup R_k$ 
16:     end if
17:   end for
18:   if  $(|\bar{\mathcal{R}}| \neq 0)$  then
19:     if  $(\sum_{\ell=1}^{|\bar{\mathcal{R}}|} O_\ell > O_z)$  then
20:        $\mathcal{R}_z \leftarrow \mathcal{R}_z - R_i$ 
21:     end if
22:   end if
23: end for
24: return  $\mathcal{R}_z$ 
```

---

We have the rule set of size  $|\mathcal{R}|$  as an input, where each rule is represented by  $\rho_i$  as its LHS and  $x_j$  as its RHS and  $|\mathcal{R}|$  denotes the number of rules in  $\mathcal{R}$ . Here, we assume that we have two classes for sake of simplicity; but we can easily extend it to more than two. In Steps 1 through 10 of Algorithm 3, we separate the rule set  $R$  into two rule sets  $\mathcal{R}_z$  and  $\mathcal{R}_{\bar{z}}$ , where they represent rules whose RHS are  $x_z$ , and  $x_{\bar{z}}$ , respectively. In addition, we calculate the weight of each rule separately.

In Steps 13 through 17 of the algorithm, for each rule  $R_i$  in  $\mathcal{R}_z$ , find all rules in  $\mathcal{R}_{\bar{z}}$  whose LHS is equal to  $\rho_i$  or its subset, and assign them to the set  $\bar{\mathcal{R}}$ . In Steps 18 through 22 of Algorithm 3, if the rule set  $\bar{\mathcal{R}}$  is not empty, then the posterior probability estimates are calculated for all the rules in this set, and then their sum is compared with the posterior probability estimate of the rule  $R_i$ ; if the former is greater, then  $R_i$  will be excluded from

$R_z$ , otherwise it remains. Steps 12 through 22 of Algorithm 3 will be repeated for each rule in  $R_z$ . The algorithm returns the attack rule set as  $R_z$ .

We apply Algorithms 2 for learning attack rules for the CIC-Friday-Morning data set, which results in 98 rules. The number of attributes in the LHS of the rules is determined by the experiments; the one which leads to high precision rate or recall rate is selected. Then using Algorithm 3, we reduce the number of rules to 5, used for the class “Bot”.

In the next chapter, we show how the rules extracted from the three methods introduced in this chapter are homomorphically evaluated to detect intrusions.

# Chapter 5

## Privacy-Preserving Intrusion Detection

In this chapter, we provide the details of our privacy-preserving rule-based intrusion detection scheme [74]. The rules are extracted from any one of the three methods as explained in Chapter 4. The scheme consists of a semi-honest protocol, where there are two parties to jointly compute a function over private inputs. In particular, we employ a secure pattern matching algorithm which privately process security data to detect a network intrusion or system anomaly. The more detailed information about participants and security models will be provided in the following subsections.

### 5.1 Participants

In the proposed protocol, there are two parties interacting with each other.

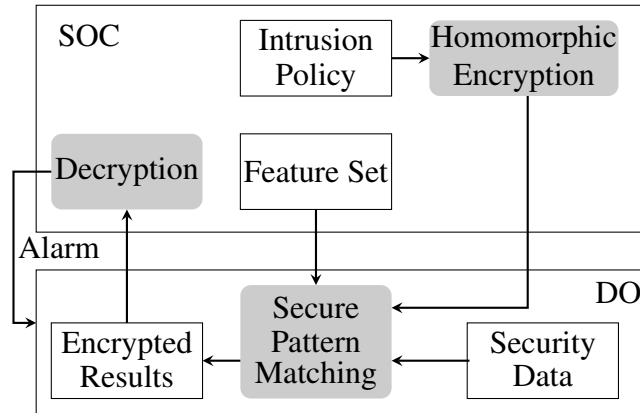
**Data Owner (DO)** owns the security data but lacks security expertise, and therefore outsources most of the activities pertaining to intrusion detection while willing to perform local computation over its security data. Due to security reasons or the private nature of the security data, DO is not willing to share it with other parties prior to proper protection. **Security Operation Center (SOC)** has expertise in security and offers intrusion detection as a service to DO. SOC makes use of detection models for discovering attacks, which are learned from past attack incidences and related data using techniques such as data mining and machine learning. Specifically in our work, the detection model is based on the *intrusion policy*, which is the set of all attack rules whose extraction is explained in Chapter 4, Due to the aforementioned reasons given in Chapter 1 (see *Secrecy of the detection model*), SOC is not willing to share detection models with other parties.

### 5.2 Semi-honest Protocol

In this section, we describe our protocol in the semi-honest model, which employs an efficient HE scheme. Ours is an interactive two-party protocol, in which we assume that

DO has sufficient computational power to perform homomorphic operations. The block diagram for the overall scheme is illustrated in Figure 5.1.

Figure 5.1: Block diagram of the overall scheme.



SOC encrypts its intrusion policy under its own public key in the HE scheme and sends the resulting encrypted intrusion policy to DO that applies it on each record of the security data. The result homomorphically encrypts the indication whether there is an intrusion. At the end of this protocol, SOC learns the indices of the offensive records, and to which attack rules these records are matched. The indices of offensive records or statistics regarding malicious records (e.g., the number of offensive records, their percentage of all processed records and the number of attack rules satisfied) are sent to DO. Other than what can be inferred from the output of the protocol, SOC does not learn anything about security data.

Besides the output (index of the offensive record), DO is given the knowledge of the feature vector  $\mathcal{A}$  and set of all values  $\mathcal{V}_i$  taken by each feature  $a_i$ . Thus, DO has *a priori* information about the detection model before protocol execution and each run of the protocol increases this information (*a posteriori* information). In subsequent sections, we give an analysis that attempts to quantify both a priori and a posteriori information and show that DO cannot fully learn the detection model if certain conditions are met.

After detection, offensive records can be put to a more detailed inspection, which usually requires participation of both DO and SOC. The first goal of the further inspection of the offensive records is possibly to classify the attacks so that an effective countermeasure or action can be taken. This is straightforward as this information comes to be known to SOC at the end of the protocol; i.e. SOC learns the attack type(s) and the index of the offensive record(s). Instead of sharing this information with DO, SOC just tells DO the countermeasure or action to be taken (e.g., terminating the connection, blocking traffic from a certain IP etc.), and perhaps the index of the offensive record. The type of countermeasure reveals only limited information about the attack type as the same countermeasure can be applied for more than one attack. Furthermore, the countermeasure can be taken even without the knowledge of the offensive records. This basically means DO

will never be able to learn the exact detection model. On the other hand, a semi-honest SOC learns only that there is an intrusion or not, if so the attack type, and the number of records pertaining to detected attacks.

In case more should be known about the attack and a specific record or records must be inspected, this stage becomes much more involved and type of action depends on the particulars of each attack class. Also, the privacy of offensive records are usually not of interest in the semi-honest model. Therefore, post-detection inspection of offensive records is beyond the scope of this work.

### 5.3 Proposed Construction

In this section, we introduce the important concepts, the protocol sketch and the algorithms used in the proposed construction. We start with *record* and *rule signatures*, which represent security data and intrusion policy, respectively, in the construction.

Normally, records in security data contain non-categorical features (e.g., numeric) as well as categorical features. Here, we use only categorical features by partitioning every non-categorical feature into at most  $t$  categories, where  $t$  is an integer appropriately chosen to achieve high accuracy in intrusion detection protocol (or depending on the feature with the maximum number categories). For instance, a category of a numeric feature may represent an interval. Thus, by abuse of notation,  $a_i = v_{i,j}$  actually means  $a_i \in v_{i,j}$  for originally numeric feature  $a_i$ , where  $v_{i,j}$  represents an interval.

Here, we use  $t$  bits to encode each feature and one-hot encoding scheme is used to encode the value of a feature in a record. Consequently, a record string is the concatenation of those  $t$ -bit strings of all features in the data set.

As categorical features are always used now, Boolean operations in the attack rules involve only test for equality (i.e., operations  $=$  and  $\neq$ ). A rule in the intrusion policy can be expressed as a string, referred as rule signature which is the same length as record signature. Although not all features are used in each rule, they are still encoded in the rule signature (e.g., using don't care values as in Boolean algebra). Therefore, the rule signature is not a binary string.

Now, we can give a sketch of our private intrusion detection protocol as follows:

1. DO generates a signature for each record (*record signature*).
2. SOC generates and encrypts a signature for each rule in the intrusion policy using its own public key and sends them to DO (*rule signatures*).
3. All attack rules are applied for each record. At the end of this phase, DO sends the encrypted detection results of each record to SOC. A detection result that encrypts 0 indicates the corresponding record is offensive; otherwise, harmless.

4. SOC checks whether there is any zero after the decryption of results for each record.
5. SOC alerts DO when there is an intrusion and inform how to proceed in the presence of an attack.

Now, we can give the algorithms for record and rule signature generation.

### 5.3.1 Record Signature Generation

A record signature is a sequence of binary digits which are generated for each record. The length of a signature is  $\ell$ , where  $\ell = d \times t$ ,  $d$  is the number of features in detection model and  $t$  is the number of bits to encode each category of a feature. We always extend  $\ell$  to the next power of two for sake of simplicity, whereby excess bits are set appropriately (i.e., to 0 in the record signature and to wildcard in rule signatures). As each feature takes one of the  $t$  possible values, to generate a record signature, DO sets the corresponding bit to 1 and the rest to 0 of the  $t$  bits representing the feature in the signature and repeats the process for each feature (see Algorithm 4).

---

**Algorithm 4** Generation of record signature

---

**Input:**  $r \in \mathcal{D}$ : A record

$\mathcal{A} = \{a_1, a_2, \dots, a_d\}$ : is a feature vector in the data set

$\mathcal{V}_i = \{v_{i,1}, \dots, v_{i,t}\}$ : List of values taken by  $a_i$

**Output:**  $s$ : record signature.

```

1:  $s \leftarrow (0)^\ell$ 
2: for  $a_i \in \mathcal{A}$  do
3:    $v \leftarrow \mathcal{D}[r, a_i]$ 
4:   for  $x \leftarrow 0$  to  $t - 1$  do
5:     if  $v = v_{i,x}$  then
6:        $j \leftarrow x$ 
7:     end if
8:   end for
9:    $s[(i - 1) \cdot t + j] \leftarrow 1$ 
10: end for
11: return  $s$ 

```

---

### 5.3.2 Rule Signature Generation

Features that are not used in an attack rule are *don't cares* as in Boolean algebra. For don't care features, the corresponding  $t$  bits in the rule signature are all set to \*, which is referred as wildcard. When a feature is not used in a rule, the corresponding  $t$  bits in the signature are all set to \*.



When the feature  $a_i$  is used in a rule and the predicate is in the form  $p_i(a_i) = (a_i = v_{i,j})$ , where  $0 \leq j \leq t - 1$ , in the corresponding  $t$  bits of the signature,  $j$ -th bit is set to 1, and the rest of the  $t - 1$  bits are set to 0. On the other hand, when the predicate is  $p_i(a_i) = (a_i \neq v_{i,j})$ , then  $j$ -th bit is set to 0, and the rest of bits are set to  $*$  in the corresponding  $t$  bits of the signature.

**Example 5.1.** Consider the intrusion policy in Example 1.4:  $P_A = R_1 \vee R_2$ , where  $R_1 = p_1$  and  $R_2 = \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4$ . Then, the rule signatures for  $R_1$  and  $R_2$  are, respectively,

$$\begin{array}{c} 001000 \dots 0 || ** \dots * || ** \dots * || ** \dots * \\ **0 * \dots * || 0 * \dots * || * 0 * \dots * || *** * 0 * \dots * \end{array}$$

where  $t = 24$ .

As the rule signature is a ternary string, we need to use two bits to represent each digit in the signature. Therefore, the rule signature can be represented as a binary matrix  $\mathbf{M}$  of dimension  $2 \times \ell$ , where each column encodes a ternary digit. The digit values 0, 1, and  $*$  are encoded with column vectors  $[0, 0]^T$ ,  $[1, 0]^T$ , and  $[1, 1]^T$ , respectively. The steps of rule signature generation are given in Algorithm 5.

### 5.3.3 Intrusion Detection Algorithm

The intrusion detection algorithm is executed by DO using a record signature (in plaintext) and the rule signatures encrypted under the public key of SOC, which is also known to DO. Thus, DO can perform homomorphic computations over the encrypted rule signature. For sake of simplicity, here we do not use a notation to indicate encrypted values. For instance, the rule signature shown as  $\mathbf{M}$  is in fact  $E_{pk}(\mathbf{M})$ , where  $pk$  is the public key of SOC. Nevertheless, the operations in the following are all applied homomorphically. Indeed, the operations are described in such a way that they can be efficiently implemented homomorphically over encrypted operands. Algorithm 6 describes the steps of intrusion detection protocol executed by DO.

---

**Algorithm 5** Rule signature generation

---

**Input:**  $R$ : Attack rule

$\mathcal{A} = \{a_1, a_2, \dots, a_d\}$ : feature vector in the data set

$\mathcal{V}_i = \{v_{i,1}, \dots, v_{i,t}\}$ : List of values taken by  $a_i$

**Output:**  $M$ : rule signature.

```
1:  $M \leftarrow (1)^{2 \times \ell}$ 
2: for  $p_i = (a_i \text{ op } v_{i,j}) \in R_i$  do
3:   if (op is =) then
4:     for  $x \leftarrow 0$  to  $t - 1$  do
5:       if  $x = j$  then
6:          $M[(i - 1)t + x] \leftarrow [1, 0]^T$ 
7:       else
8:          $M[(i - 1)t + x] \leftarrow [0, 0]^T$ 
9:       end if
10:    end for
11:   else
12:      $M[(i - 1)t + j] \leftarrow [0, 0]^T$ 
13:   end if
14: end for
15: return  $M$ 
```

---

---

**Algorithm 6** Intrusion detection algorithm

---

**Input:**  $M_i$ : Signatures of rules  $R_i \in P$  for  $i = 0, \dots, u - 1$

$u$ : number of rules in  $P$

$s$ : Signature of record  $r$

$\ell$ : Signature length

**Output:**  $\xi$ : Detection result for record  $r$ .

```
1:  $\xi \leftarrow (0)^{1 \times u}$ 
2: for  $i \leftarrow 0$  to  $u - 1$  do
3:   for  $j \leftarrow 0$  to  $\ell - 1$  do
4:      $\tau[j] \leftarrow (s[j] \vee M_i[1, j]) \oplus M_i[0, j], \tau = (0)^{1 \times \ell}$ 
5:   end for
6:   for  $x \leftarrow 0$  to  $\log_2 \ell$  do
7:      $\tau \leftarrow \tau + (\tau \lll 2^x)$ 
8:   end for
9:    $\chi \leftarrow (0)^{1 \times \ell}$ 
10:   $\chi[0] \xleftarrow{\mathbb{R}} \mathbb{Z}_p$ 
11:  for  $j \leftarrow 0$  to  $\ell - 1$  do
12:     $\tau[j] \leftarrow \tau[j] \cdot \chi[j]$ 
13:  end for
14:   $\tau \leftarrow (\tau \ggg i)$ 
15:   $\xi \leftarrow \xi + \tau$ 
16: end for
17: return  $\xi$ 
```

---

Algorithm 6 takes rule signatures  $M_i$  (in encrypted form) and record signature  $s$  and outputs an encrypted result vector  $\xi$ , which is sent to SOC for decryption. The vector  $\xi$  consists of  $u$  integers and encodes the result of Algorithm 6 that homomorphically executes the policy  $P$  over record  $r$ . Each element of  $\xi$  holds an integer value corresponding to one rule in  $P$ . A zero value in a vector element simply means the corresponding attack rule is satisfied (i.e., an attack is detected); otherwise, it contains a random integer.

The algorithm runs the outer **for** loop (see Step 2 of Algorithm 6), in each iteration of which one attack rule is checked and the result is saved in the result vector  $\xi$  (see Step 15). In Step 3 of the algorithm, the **for** loop implements string matching of the record and rule signatures, where wildcard digits in the rule signature match to any bit in the record signature eliminating their effect in the matching result. Here a match is used to indicate that a feature value in the record signature satisfies the predicate in the rule signature, in which it is used. When this happens, the corresponding  $t$  bits in  $\tau$  evaluate to 0; otherwise a non-zero value. When all predicates are satisfied we have  $\tau = (0)^{1 \times \ell}$ .

Then, the **for** loop in Step 6 computes a fast addition of all bits of  $\tau$  in  $\log_2 \ell$  steps and

the sum is accumulated in the leftmost bit of  $\tau$ ; namely in  $\tau[0]$ . In case of all predicates are satisfied in the rule, we will have  $\tau[0] = 0$ . Otherwise, the bits of  $\tau$  have non-zero values, which reveal the number of predicates that are not satisfied. Since this can leak information about the record  $r$  to SOC when  $\tau$  is decrypted,  $\tau[0]$  is multiplied by a uniformly random number (Step 10) and the rest of the bits are reset to 0 (see the **for** loop in Step 11). Thus, only  $\tau[0]$  keeps the result, which is saved in a slot of another ciphertext  $\xi$  to save space and bandwidth (see Step 14 and 15).

What SOC learns after the decryption of  $\xi$  is whether any rule in attack policy  $P$  matches with the record or not. In case of a detection, SOC learns also the matching attack rule or rules, which will help SOC classify the attack and decide on the proper countermeasure or action. To avoid extra shift and multiplication operations, we extend the detection result for each record from  $u$  slots to  $\ell$  slots. Consequently, the detection result for only  $N/\ell$  records can be stored in each ciphertext, where each  $\ell$  slots belongs to one record. There are  $u$  rules in a policy and therefore the detection result for a single record takes the first  $u$  slots in the corresponding  $\ell$  slots of the record in the resulting ciphertext, which will be sent to SOC after Algorithm 6 is executed by DO. In the ciphertext, SOC only considers the first  $u$  slots for each record and ignore the  $\ell - u$  slots. In Chapter 7, we provide experimental results for real data sets that also show how many detection results can be encrypted in the same ciphertext.

Algorithm 6 can be efficiently computed since the depth of the homomorphic computation is very low. Homomorphic computations in Step 4 involve relatively simple operations. For instance, the bit-wise logical-OR operation of  $s$  and  $M$  is relatively easy as the former is not encrypted. The following exclusive-OR operation requires only one homomorphic multiplication. Similarly, the multiplication in Step 12 is efficient since only  $\tau$  is encrypted. The homomorphic shift operations in Step 7 and Step 14 are not expensive while the cost of homomorphic addition in Step 15 is negligible. Finally, iterations of the outer **for** loop are independent of each other and therefore can be performed in parallel.

The timing results and comparison with other works are provided in Chapter 7 to show the efficiency of the proposed construct.

# Chapter 6

## Privacy and Security Arguments

In this section we provide privacy requirements as definitions and then argue that the proposed protocol addresses these requirements.

**Definition 6.1** (Ciphertext Indistinguishability). *Given a pair of arbitrarily chosen messages  $m_0$  and  $m_1$ , and the ciphertext  $E(m_b)$  for a uniformly random  $b \leftarrow \{0, 1\}$ , a polynomially bounded adversary  $\mathcal{A}$  that has access to encryption oracle, has negligible advantage in guessing  $b$  correctly; i.e.,  $Adv_E(\mathcal{A}) = 2|Pr(\mathcal{A} \text{ wins}) - 1/2|$ .*

Definition 6.1 is essentially IND-CPA that secures the messages exchanged between DO and SOC against third parties. It also protects SOC's detection model against DO.

**Definition 6.2** (Predicate Privacy). *Given an attribute  $a_i$ , its value set  $\mathcal{V}_i$  and a security data set  $\mathcal{D}$  with  $|\mathcal{D}| = n$ , it is difficult to guess a predicate  $p_i(a_i) = (a_i = v_{i,j})$  used in a detection rule and the difficulty is affected by the entropy of  $a_i$  in  $\mathcal{D}$ .*

Definition 6.2 basically states that one can always guess predicates used in detection model as an adversary is given all the attributes and their value sets, and naturally collects and observes a certain amount of security data, which has usually biased statistics. That a feature takes certain values more often than others and/or never takes some values can give certain advantage to adversary for guessing, which is still difficult if the entropy is sufficiently high. We suggest to use Shannon entropy as a measure as to how much of detection model is leaked to DO when features and their value sets are made known to DO before the execution of the protocol. Once the protocol runs, DO can gain more information about predicates used in detection model via the protocol outcome. However, DO cannot learn the predicates in a specific rule as the outcome does not specify any rule.

**Definition 6.3** (Rule Privacy). *A privacy-preserving intrusion detection provides rule privacy, if DO cannot construct attack rules by aggregating the offensive records.*

**Definition 6.4** (Model Privacy). *A privacy-preserving intrusion detection provides model privacy if malicious DO, cannot construct the model after sending polynomially bounded number of queries to SOC.*

**Definition 6.5** (Data Privacy). *A privacy-preserving intrusion detection provides data privacy if SOC cannot reveal information about DO’s data by changing the rules adaptively and observing the results.*

**Definition 6.6** (Access Control). *A privacy-preserving intrusion detection provides access control if an unauthorized SOC cannot impersonate an authorized SOC and access the detection results.*

Here we argue that the proposed protocol addresses the defined privacy requirements. We assume that the set of features  $\mathcal{A}$ , and sets of all values ( $\mathcal{V}_i$ ) each feature  $a_i$  takes are known to both DO and SOC. We analyze the security of the protocol against an adversary that can access all communication between any two parties. Moreover, adversary may have some a-priori information on security data or data model.

**Theorem 6.1.** *The proposed privacy-preserving intrusion detection protocol provides ciphertext indistinguishably for detection rules, sent by SOC and detection results by DO in accordance with Definition 6.1.*

*Proof.* The detection rules and results are encrypted under the public key of SOC using lattice-based probabilistic encryption algorithm, which is IND-CPA secure. Therefore, no adversary (including DO against detection rules) wins the security game outlined in Definition 6.1 except for negligible probability.  $\square$

**Theorem 6.2.** *The proposed privacy-preserving intrusion detection protocol provides predicate privacy in accordance with Definition 6.2.*

*Proof.* We prove that it is difficult to deduce a predicate used in any of the detection rules as every attribute has a certain entropy even after the discretization operation. The entropy for a feature  $a_i$  is computed as

$$E(a_i) = - \sum_{v_{i,j} \in \mathcal{V}_i} p(v_{i,j}) \log_2 p(v_{i,j}) \quad (6.1)$$

where  $\mathcal{V}_i = \{v_{i,1}, \dots, v_{i,t}\}$  is the set of all possible values such that  $a_i \in V_i$ .

In data discretization the number of distinct values for a continuous feature is reduced by partitioning its range into a finite set of disjoint intervals. We select an unsupervised binning method to transform numerical features into categorical counterparts where the target (class) information is not used. There are two binning methods as *equal width* and *equal frequency binning*. In equal width binning, the data is divided into  $t$  intervals of equal size. The width of each interval is  $\frac{(max-min)}{t}$ , where *max* and *min* are maximum and minimum values in the corresponding interval, respectively. In equal-frequency binning, the data is partitioned into  $t$  groups, each of which contains approximately same number of values.

Feature name	Normal		Discretized	
	Entropy	Distinct value	Entropy	Distinct value
AppName	2.72	68	2.72	68
Total source bytes	8.52	1210	4.56	24
Total dest bytes	8.92	1760	4.55	24
Total dest packets	4.64	207	3.98	24
Total source packets	4.27	171	3.72	24
Direction	0.80	4	0.80	4
SourceTCPFlagDesc	2.27	17	2.27	17
DestTCPFlagDesc	2.006	18	2.006	18
Protocol Name	0.71	4	0.71	4
Source port	10.84	2882	4.57	24
Destination port	2.58	291	2.22	24

Table 6.1: Comparison of entropy of numerical variables and discretized variables in ISCX-Saturday data set

In Table 6.1 we list the entropy values for 11 features computed for the ISCX-Saturday test data set before and after discretization, where equal-frequency binning method is chosen as binning method and  $t$  is set as 24. The aim is to show how much entropy is lost due to the discretization of numerical features. The entropy for each feature in the test data set is calculated using Equation 6.1 and enumerated in the column labeled with “Entropy”. The “Distinct value” column lists the number of distinct values each feature takes in the records of the data set.

The results show that the entropy of numerical features is decreased only by 29.57% on average after discretization, and the entropy of categorical features are not affected. In order to further increase the total entropy we can i) increase the number of intervals,  $t$ , in discretization and ii) introduce additional attributes that are not used in the detection model.

□

**Theorem 6.3.** *The privacy-preserving intrusion detection protocol provides rule privacy in accordance with Definition 6.3.*

*Proof.* SOC encrypts an intrusion policy under its own public key in the HE scheme and sends the resulting ciphertext to DO, which homomorphically applies it on each record of security data. The result homomorphically encrypts the indication whether there is an intrusion. Having decrypted the ciphertext, SOC learns the indices of the offensive records, and which intrusion rules are violated. While the indices of offensive records or statistics regarding malicious records are sent to DO, the information about, as to which attack rule each offensive record violates, is hidden. SOC can tell DO how to react to each violation, which does not necessarily leak information about the attack rule itself as

common responses to many attacks are applied in many cases.

DO knows the number of rules as each rule is applied individually. The number of rules can be hidden by introducing dummy rules at the expense of an overhead in execution times.  $\square$

**Theorem 6.4.** *The privacy-preserving intrusion detection protocol provides model privacy in accordance with Definition 6.4.*

*Proof.* A malicious DO may want to reconstruct the detection model (e.g., decision tree) (or an approximation of it) by exhaustively trying all possible queries. Dimension  $d$  of the feature vector and the number of bits  $t$  (which is needed to encode each attribute in the feature vector) are known to DO (a priori information), which is, therefore, able to generate  $t^d$  different queries, whereby each time different values for attributes in the feature vector are selected. The response sent by SOC to DO may disclose information about the detection model (a posteriori information). In order to render exhaustive search by DO infeasible, larger values of  $d$  and  $t$  can be adopted by introducing unused attributes and using unnecessarily high number of distinct values in discretization process. However, there is a trade-off between detection time and security. The larger the value of the  $d$  and  $t$ , the larger the signature size, and slower detection time; however stronger security against malicious DO. In the CIC-Friday-Morning data set, where the number of features is  $d = 50$ , and  $t = 10$ , it takes about 77 seconds to generate detection results for 4010 records. A malicious DO that wants to reconstruct the detection model, may have to submit  $10^{50}$  queries in the worst case scenario. It takes about  $1.92 \times 10^{48}$  seconds to generate detection results for  $10^{50}$  records, which is  $6.09 \times 10^{40}$  years. As a result, it will be impractical for a malicious DO to recover the detection model in full in a reasonable amount of time.  $\square$

**Theorem 6.5.** *The proposed privacy-preserving intrusion detection protocol partially provides data privacy against malicious SOC in accordance with Definition 6.5.*

*Proof.* A semi-honest SOC will learn only whether a record matches to a detection rule or not after DO runs a legitimate detection model on data records. This does not give much information about the feature values in a record. However, malicious SOC may generate false detection rules to extract more information about a feature in a record. For example, SOC can form a rule, in which there is one predicate testing if a feature takes on a certain value. This is especially dangerous for categorical features that take two values. SOC may even try to reconstruct a target record by aggregating the results to the false detection rules. Even in this case, however, malicious SOC can gain only limited information about the record. In particular, the dimension of the feature vector ( $d$ ), and the number of categories of a feature ( $t$ ), play important roles in limiting the SOC's reconstruction capability of a target record. For instance, SOC needs to send  $d \times t$  rules to fully reconstruct a target



record. On the other hand, the number of the detection rules is generally much lower than  $d \times t$ . For example, in the CIC-Friday-afternoon data set, the number of rules is 19 while  $d \times t = 500$ . As these 19 rules are applied once to a record, malicious SOC will learn only very limited information about the features. Increasing the number of features and their categories renders the re-construction of a target record more difficult. DO can refuse to evaluate the rules or send the resulting ciphertext to SOC if number of rules are relatively high in comparison with  $d \times t$ . In addition, data discretization used in our protocol provides further protection against malicious SOC since not the exact values of many features of a record, but their ranges are used. Consequently, malicious SOC will learn the interval of values for many features.  $\square$

**Proposition 6.1.** *DO can hinder SOC from reconstructing a target record by applying a random permutation on data records for each detection rule.*

As pointed above, via malicious rules sent to DO, SOC can reconstruct a target record (probably partially) by aggregating the results returned by DO. In our solution, DO generates a random permutation of  $y$  records for each SOC rule,  $R_i$ , where there are  $u$  rules. DO stores these random permutations in a matrix  $\mathbf{P}$ , where there are  $u$  rows and  $y$  columns. DO applies each detection rule on a record almost the same way as in Algorithm 6. The only difference is that the results of  $y$  records for each rule are now permuted using the corresponding permutation and placed accordingly in the slots of final result  $\xi$ , which is sent to SOC. It is now impossible for SOC to reconstruct a target record by aggregating the results to the detection rules as SOC observes random indices for each detection rule. After having decrypted the results, SOC specifies the indices of slots evaluating to zero, and sends them to DO. Using  $\mathbf{P}$  DO finds the indices of the records which are classified as attacks.

**Example.** Suppose we have two detection rules  $R_0$  and  $R_1$ , and eight records  $(r_1, r_2, \dots, r_8)$  to be classified as either normal or attack. First DO permutes the records randomly for each detection rule. Suppose random permutations  $\{4, 1, 8, 5, 2, 6, 7, 3\}$  and  $\{5, 2, 4, 8, 7, 3, 6, 1\}$  are selected for the  $R_0$  and  $R_1$  and permutation matrix is constructed as

$$\mathbf{P} = \begin{bmatrix} 4 & 1 & 8 & 5 & 2 & 6 & 7 & 3 \\ 5 & 2 & 4 & 8 & 7 & 3 & 6 & 1 \end{bmatrix}$$

DO homomorphically applies the rule  $R_i$  on each record using the order in the  $i$ -th row of  $\mathbf{P}$ . Consequently, the result of  $r_4$  for the rule  $R_0$  will be placed in the first slot of  $\xi$ ,  $r_1$  in the second slot, and so on. Similarly, the permutation in the second row of  $\mathbf{P}$  is employed for  $R_1$ .

Then, the encrypted result  $\xi$  is sent to SOC that decrypts and finds out indices of slots that contain 0, which indicates the corresponding attack rule is fired. But, he cannot link a record that fires more than one rule as a different permutation is used for each rule.

SOC sends these indices of the slots that fire a rule without specifying the mapping between the rules and indices. Suppose the following is the index matrix corresponding to the slots of  $\xi$

$$\mathbf{I} = \begin{bmatrix} 0 & 2 & 4 & 6 & 8 & 10 & 12 & 14 \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 \end{bmatrix}.$$

Assume also SOC observes zeros in the indices  $\{0, 1, 6, 14\}$  and sends them to DO. Using the permutation matrix  $\mathbf{P}$ , DO understands that the records  $r_3, r_4$ , and  $r_5$  fire attack rules. Naturally, DO now also learns that the record  $r_5$  fires two rules, which leaks more information to DO than previously claimed in this dissertation. By increasing the number of rules and/or introducing dummy rules SOC can partially protect its rules against DOC's query attack. On the other hand, SOC may prefer not sending the indices that fire specific rules to DO if the offensive records are not needed to be known. After all, what is important is whether a rule is fired or not. SOC just instructs DO what to do in case specific attack rules are fired.

For  $y$  records, there are  $y!$  different random permutation, and if  $u$  is much less than  $y!$  (can be always made true by choosing sufficiently many records), there is no chance for SOC to correctly guess the permutations selected by DO. Since SOC will observe different indices for all those rules, it can not determine whether two different rules are satisfied by two different records or by the same record. The detection time is almost the same with the timings in Table 7.3, suggesting that the proposed solution is applicable in case of malicious SOC.

**Theorem 6.6.** *The privacy-preserving intrusion detection protocol provides access control in accordance with Definition 6.6.*

*Proof.* The authorized SOC has a public-private key pair and its public key is assumed to be made known to DO by secure means. Thus, DO refuses to use any other public key other than the one of the authorized SOC during homomorphic operations. Consequently, only the authorized SOC can decrypt ciphertext sent by DO and learns the detection results. □

# Chapter 7

## Experiments

To demonstrate the effectiveness of the proposed privacy-preserving intrusion detection protocol, we implemented it using Simple Encrypted Arithmetic Library- SEAL v2.2 [68] that employs the FV homomorphic encryption scheme [9]. Then we applied it to real data sets and reported its performance using various metrics such as execution time, bandwidth, and accuracy. To compare its performance against similar proposals reported in other studies, we applied our protocol on a data set extracted from ISCX data set with the same number of features and depth as the other works use, as there is no work in the literature that process security data as such. We choose three well known machine learning classification algorithms (decision tree, naïve Bayesian, and neural network to analyze a test data and extract the attack models. In all the experiments in this dissertation, 70% of the records are used for training while 30% for testing.

### 7.1 Experimental Setup

In our experiments, we adopt mainly ISCX 2012 IDS [5] and CIC IDS 2017 data sets [51] as benchmark to report the performance metrics of the proposed protocol. The ISCX 2012 IDS data set captures network activity (normal and malicious) of seven days generated in a controlled test bed by the Information Security Centre of Excellence (ISCX) at the university of New Brunswick<sup>1</sup>. Similarly, the CIC IDS 2017 data set captures network activity of five days in a test bed, designed by Canadian Institute for Cybersecurity (CIC) at University of New Brunswick. In our experiments we used the Saturday data set of the ISCX 2012 IDS (henceforth ISCX-Saturday). In CIC IDS 2017 data set, we used data from Friday morning and Friday afternoon. Friday morning data (CIC-Friday-Morning) includes network packets during Botnet attacks while Friday afternoon data (CIC-Friday-Afternoon) is collected during DDoS attacks.

---

<sup>1</sup>Available at <https://www.unb.ca/cic/datasets/ids.html>

### 7.1.1 Feature Selection

In machine learning and statistics, feature selection is the process of selecting a subset of relevant features for use in model construction. The main reason for employing feature selection is that data sets usually contain some features that are either redundant or irrelevant, and can thus be removed without incurring significant information loss. We use attribute evaluation methods, provided by WEKA to optimize feature selection process; however our algorithm is independent of any such tool, and works with features selected by any means. We essentially used Information Gain (IG) attribute evaluation method [75], which assesses the significance of a feature with respect to a class. For instance, in our experiments with the CIC-Friday-Afternoon data set, we reduce the number of features to 25, by considering only those features of ranks being more than 0.0387. Some of the features such as “Flow ID”, “Source IP”, “Destination IP”, and “timestamp” are represented by overly many categories that can render our method approach inefficient, if not infeasible. For example, in the CIC-Friday-Morning data set, “Flow ID” and “timestamp” features have 1019 and 241 categories, respectively. While the former feature can easily be eliminated due to its low rank, the latter has negative impact on detection performance if eliminated. In such cases, we keep the feature and apply a technique to reduce the number of categories without adverse effect on the detection performance; e.g., we can map all its possible values to numerical values, which can be discretized to  $t$  categories.

Features such as “source IP”, “destination IP”, “startDateTime”, “stopDateTime”, and “Flow ID” considered as categorical (by WEKA as well as in our method), take on values in a wide range resulting in overly many categories, and thus larger signature size. To prevent signature size from becoming too large, we remove the features if they do not adversely affect the detection rate. However, if these features are essential in detection performance, we show here, when applicable, how the number of categories is reduced. As an example “timestamp” is a categorical feature, which takes values in DD:MM:YYYY HH:MM format. Since existing data sets are classified depending on the date, all records in a sample data set has the same date. Thus, we remove the date part from the format; and for the rest, HH:MM, we remove “:”; then all values are numerical values which can be discretized to any desirable number of categories.

## 7.2 Evaluation Metrics

In its simplest form, intrusion detection can be formulated as a two-class problem. Thus, a record in security data is classified as benign (normal) or malicious (attack). When a record is classified as attack, IDS generates an alarm. Generally speaking IDS is usually not perfect and creates false alarms or misses some attacks. To evaluate the performance

of IDS, we measure four values: i) **true positive (TP)** is the number of malicious records that are correctly classified as attacks; ii) **false positive (FP)** is number of benign records classified as attacks; iii) **false negative (FN)** is the number of malicious records classified as normal; iv) **true negative (TN)** is the number of benign records classified as normal. Based on these measurements in our experiments, the performance of our protocol is evaluated using four metrics:

1. **Accuracy Rate (AR):** The ratio of the number of correctly classified records to the number of all records

$$AR = \frac{TP + TN}{TP + TN + FP + FN} \quad (7.1)$$

2. **Precision:** The ratio of the number of records correctly classified as attack to the total number of alarms generated by IDS

$$Precision = \frac{TP}{TP + FP} \quad (7.2)$$

3. **Recall (detection rate):** The ratio of the number of records correctly classified as attack to the total number of attacks

$$Recall = \frac{TP}{TP + FN} \quad (7.3)$$

4. **False Alarm Rate (FAR):** The ratio of the number of false alarms to the number of correctly classified records

$$FAR = \frac{FP}{TN + FP} \quad (7.4)$$

### 7.3 Results and Discussions

All implementations regarding training the data, extracting attack rules, and generating signatures are written in Java programming language using WEKA library [76]. The proposed privacy-preserving intrusion detection protocol is implemented in C++. We set the security level to at least 128 bit in the SEAL library [68].

The experiments are divided into two parts: the first part uses only the decision-tree-based intrusion detection model, whereas the second part reports the comparison of the three rule-based intrusion detection models.

Data set	Random sample size	Attack(#)	$d$
ISCX-Saturday	13775	1017	15
CIC-Friday-Morning	13371	137	84
CIC-Friday-Afternoon	13543	2509	84

Table 7.1: Parameters used by WEKA : Attack(#) is the number of malicious records in the random sample data set,  $d$  is the dimension of the feature vector.

### 7.3.1 Decision-tree based model results

We conduct these experiments on a machine with Intel Xeon 6 Core running at 3.2 GHz processor and 8 GB of RAM. We take advantage of the technique provided in SEAL, which is called batching in the homomorphic encryption literature. The choice of encryption parameters (ring degree  $N$ , coefficient modulus  $q$ , and plaintext modulus  $p$ ) significantly affects the performance, capabilities, and security of the encryption scheme. In our experiments, the ring degree  $N$  is set to 4096 and the plaintext and ciphertext modulus are selected automatically by the SEAL library depending on  $N$  and the security level.

In order to determine the accuracy of our provided protocol, we compared the detection results obtained from WEKA over the original data set (i.e., the data set is not encrypted) with the results obtained from our protocol. The parameters used for the three data sets are listed in Table 7.1. Since there are large numbers of records in each data set, we pick random samples from each data set and place them in either training or test sets. The first row of the table pertains to the ISCX-Saturday data set, where there are 13775 records in the data set. While number of malicious records in the data set is 1017, the dimension of the feature vector is 15. Also, the second and third rows of the table enumerate the parameters for CIC-Friday-Morning and CIC-Friday-Afternoon, respectively.

In Table 7.2 we list some of the parameters used or adapted while applying our protocol on the three data sets. In the second column,  $\mathcal{T}_{IG}$  stands for the IG threshold for deleting irrelevant or low impact features. Features such as “source IP” and “destination IP”, “startDateTime”, “stopDateTime”, and “appName” in ISCX-Saturday, considered as categorical (by WEKA as well as in our method), take on values in a wide range resulting in overly many categories. In addition, features with negligible ranks and no negative impact on detection performance, can be removed if they lead to long signatures. If they can be accommodated in unused message slots of the underlying FE scheme, we can keep them as extra features that can prove to be beneficial for detection model privacy (see Chap-

Data sets	$\mathcal{T}_{IG}$	$d$	$t$
ISCX-Saturday	-	10	24
CIC-Friday-Morning	0.010	50	10
CIC-Friday-Afternoon	0.139	20	25

Table 7.2: Parameters used in our protocol:  $\mathcal{T}_{IG}$  is the threshold for deleting irrelevant features,  $d$  is the dimension of the feature vector after feature elimination,  $t$  is the number of categories of the feature with the maximum number of categories.

ter 6). The second column in Table 7.2 represents the dimension of the feature vector after feature elimination.

After feature selection, a remaining numerical feature,  $a_i$  is discretized to  $t$  categories, where  $t$  is the number of categories of the feature with maximum number of categories. For the ISCX-Saturday data set in Table 7.2,  $t$  value (24) is already determined by the number of categories of originally categorical features. For the CIC-Friday-Morning and the CIC-Friday-Afternoon data sets,  $t$  is set to the values (10 and 25, respectively) used in discretization of numerical features.

In the first row of Table 7.2, a threshold value  $\tau_{IG}$  is not used as the number of features already is small. Only five features are deleted due to the fact that they result in overly many categories. In the second row, four features result in many categories and therefore are deleted. In addition, to delete low impact and irrelevant features, threshold value of 0.010 is used; consequently, the number of features is decreased to 50. This optimization would lead to gain in execution time as the length of the record signature is decreased to 500 ( $= d \times t$ ). Since our method requires that a record signature always use number of slots that is a power of 2, so the next power is 512 for 500 instead of 1024 for 800<sup>2</sup>.

Table 7.3 compares the performance of the proposed protocol against the best results obtained by WEKA using correctness metrics AR, Precision, Recall, and FAR. The row  $\ell$  lists the lengths of the signature for each data set in our protocol. The rows ‘‘Test size’’, ‘‘Model size’’, and  $u$  specify the total number of records used for testing, the total number of nodes (decision + leaves nodes) in the tree, and the number of attack rules in the model, respectively. The row ‘‘Detection time’’ lists the execution times to apply all attack rules in our protocol for the data sets while the last row lists the parallel execution times when 12 threads are used in the test computer. The results in the table show that our protocol has very high correctness performance and are only slightly worse than WEKA results. For time performance, our protocol can check about 4000 records in at most 27.73 s on an off-the-shelf desktop computer.

<sup>2</sup>It is the signature length before low impact and irrelevant features are deleted.

Data set	ISCX-Sat		CIC-Fri-Mor		CIC-Fri-After	
	WEKA	Ours	WEKA	Ours	WEKA	Ours
AR(%)	99.88	99.80	100	99.97	99.95	99.70
Precision (%)	99.67	99.66	100	97.67	100	98.68
Recall (%)	98.69	97.71	100	100	99.73	99.73
FAR (%)	0.026	0.026	0	0.025	0	0.30
$\ell$	-	256	-	512	-	512
Test Size	4131	4131	4010	4010	4064	4064
Model Size	23	23	5	25	25	95
$u$	7	6	2	6	1	19
Detection time (s)	-	38	-	78	-	265
Detection time Parallel (s)	-	3.73	-	7.79	-	27.73

Table 7.3: Comparison of detection results by WEKA and the proposed protocol over three data sets.

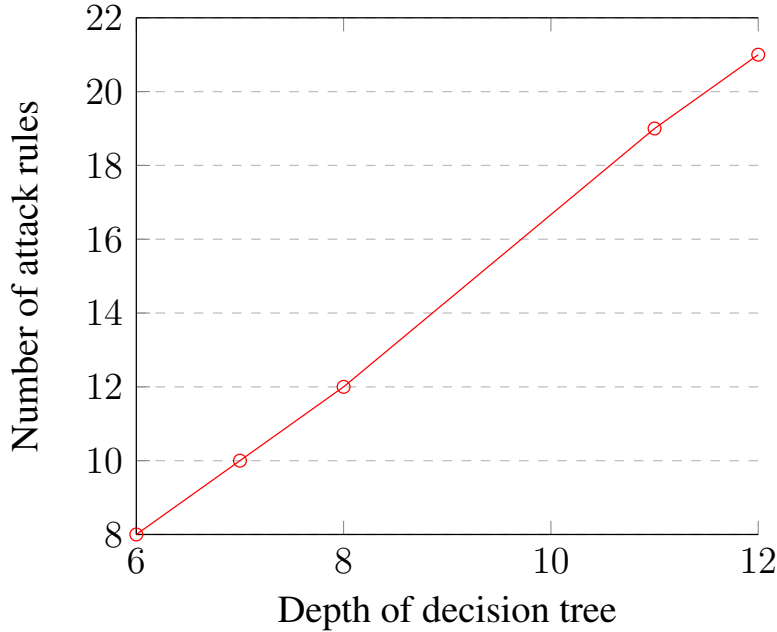
Data set	$d$	$d_t$	$d_n$	Method	Secur Level (bits)	Computation (s)		Bandwidth (KB)
						Client	Server	
Nursery	8	4	4	[64]	80	1.579	0.798	2639.0
				[66]	128	0.113	0.126	101.7
ISCX	8	4	4	Ours	128	0.05	0.084	640

Table 7.4: Parameters:  $d$  is the dimension of the feature vector,  $d_t$  is the depth of the decision tree model,  $d_n$  is the number of decision nodes.

To evaluate the time performance and the communication efficiency of the proposed protocol, we compare it against the protocol in [64] on the Nursery data set from the UCI Machine Learning Repository [67], and the protocol in [66] on the tree with the same depth and the number of decision nodes as in [64], as they are the only studies that use similar approach to ours. Since the decision tree used in [64] for the Nursery data set is not precisely specified, we used a decision tree derived from the ISCX-Saturday data set with the same feature vector size, depth, and the number of decision nodes as in [64, 66]. In our benchmarks, we measure the computation time and the total number of bytes exchanged between SOC and DO for detection of a record. The results are summarized in Table 7.4 and show that despite running at high security level (128 bits), our protocol performs better in terms of computation time compared to the protocols in [64, 66]. The execution time of our protocol is superior in both server and client computers (SOC and DO computers in our protocol, respectively). The total bandwidth in our protocol is computed using two attack rules. Thus, SOC sends about  $2 \times 128$  KB for each rule; DO sends



Figure 7.1: Variation of the number of attack rules ( $u$ ) with respect to the decision tree depth in CIC-Friday-Afternoon data set.



approximately 128 KB as the result. Although the total bandwidth for a query in [66] is less than the total bandwidth for a record in our protocol, our protocol scales better than the one in [66]. Firstly, the bandwidth of the protocol in [66] grows exponentially in depth of the tree while in our protocol SOC bandwidth depends on the number of attack rules ( $u$ ), and DO's bandwidth depends on the signature length ( $d \times t$ ) and the number of records. Furthermore the bandwidth remains constant for every  $N/\ell$  records. Also attack rules are sent once by SOC so long as the detection model is not updated. As the detection model is not expected to change frequently, the bandwidth used by DO dominates the communication.

We perform a series of experiments on the CIC-Friday-Afternoon data set. In Figure 7.1, we compare the variation of number of attack rules with respect to the decision tree depth, where we derived decision trees with different depths from the CIC-Friday-Afternoon data set. The result shows that the number of attack rules increases linearly in depth of the tree. As the execution time and the bandwidth depend on the number of the attack rules, we can say that our protocol scales well for deep decision trees and parallel computing platforms benefit our protocol.

In Figures 7.2 and 7.3, the computation time and bandwidth at DO and SOC are measured by applying the detection protocol over only one record depending on the number of attack rules ( $u$ ). All variables of the feature vector are categorical, where the number of categories is  $t = 25$ . The dimension of the feature vector is  $d = 20$ , therefore signature size is set as  $\ell = 512$ . The important observation here is that the computation times of both DO and SOC grow linearly in the number of attack rules. The amount of data sent

from DO to SOC (DO Upload in the figure) does not change for a single record as one ciphertext encrypts all results. On the other hand, the communication from SOC to DO (SOC Upload) grows linearly in the number of attack rules. Recall that SOC sends attack rules only when they are updated and SOC bandwidth is not affected by the number of records.

Figure 7.2: DO and SOC computation time (excluding network communication) for one record in the CIC-Friday-Afternoon data set with respect to the number of attack rules with  $t = 25$  and  $\ell = 512$ .

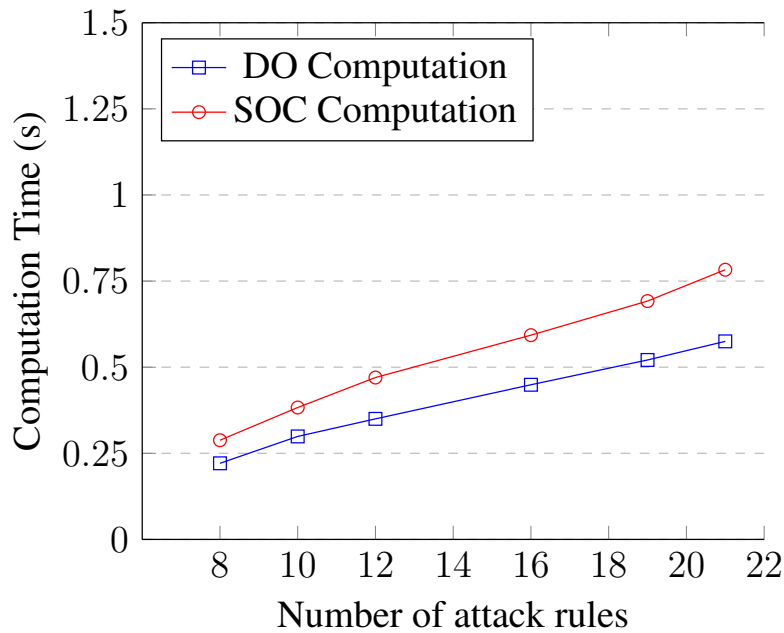
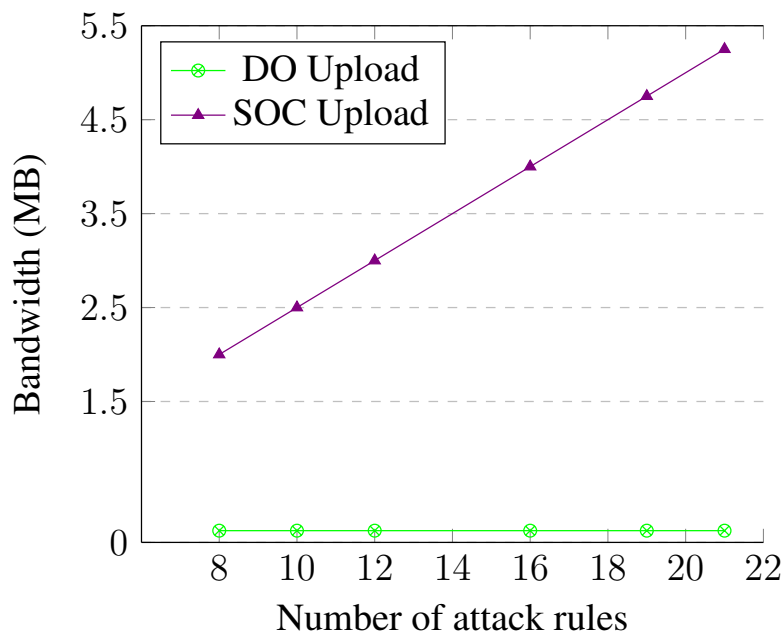


Figure 7.3: DO and SOC communication uploads for one record in the CIC-Friday-Afternoon data set with respect to the number of attack rules with  $t = 25$  and  $\ell = 512$ .



Figures 7.4 and 7.5 summarize the results of the experiments conducted on the CIC-Friday-Afternoon data set, which contains 4064 records. Simply speaking the figures illustrate the effect of increasing the dimension of the feature vector ( $d$ ) on detection time (the DO computation time), and DO upload, respectively, where each feature is discretized into  $t = 25$  categories, and the number of attack rules is  $u = 19$ . As the dimension of the feature vector affects the signature size, we expect a linear increase in computation time with respect to feature vector size; a behavior, which is captured in Figure 7.4. The computation time increases in jumps as the signature size does not change for certain intervals of  $d$ . For instance, The signature size is the nearest power of 2 to  $d \times t$  for  $6 < d < 10$  and  $t = 25$ ; i.e.,  $\ell = 256$ . Similarly, the signature size remains as  $\ell = 512$  when  $11 < d < 20$  in the figure. Consequently, the computation time remains constant for the same signature size independent of the dimension of the feature vector. In the figure, we also report the computation time of parallel implementation, where there are 12 threads running on six CPU cores. As a result, we have  $10\times$  improvement in detection time, which implies good scalability.

Figure 7.4: DO computation time for 4064 records in CIC-Friday-Afternoon data set depending on dimension of the feature vector with  $t = 25$  and  $u = 19$ .

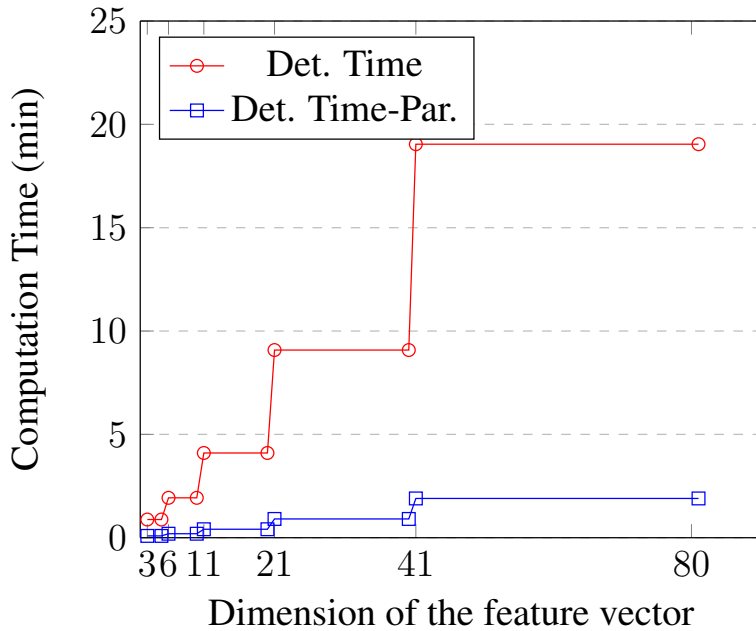
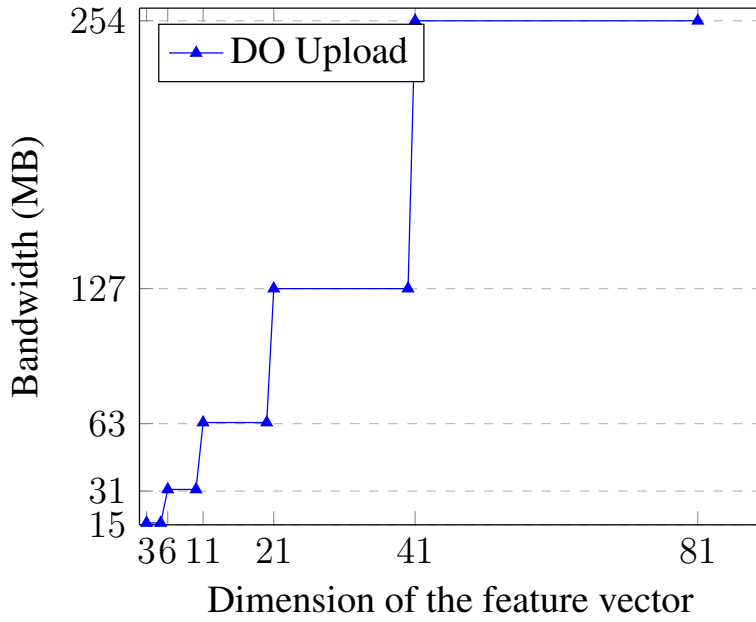


Figure 7.5: DO upload for 4064 records in CIC-Friday-Afternoon data set depending on dimension of the feature vector with  $t = 25$  and  $u = 19$ .



The amount of data sent from DO to SOC depends on the number of the records and the signature size. Thus, the dimension of the feature vector affects the DO bandwidth. Each ciphertext sent by DO to SOC is 128 KB, and contains the detection result for  $\frac{N}{\ell}$  records. Therefore, the total number of ciphertexts that must be sent to SOC is  $\frac{n}{N/\ell}$ . DO upload also increases in jumps as the signature size does not change for certain intervals of  $d$ . Figure 7.6 and 7.7 illustrate the effect of increasing the number of records ( $n$ ) on DO computation time (i.e. detection time in the figure) and DO uploads while model size remains unchanged. The experiments are conducted on the CIC-Friday-Afternoon data set. The signature size is set as  $\ell = 512$ . DO upload will not be changed for each  $N/512$  records. We also measure the detection time using parallelism where there are 12 threads in our system. As a result, we have  $10\times$  improvement in DO computation time.

Figure 7.6: DO's computation time depending on number of records in CIC-Friday-Afternoon with  $\ell = 512$ .

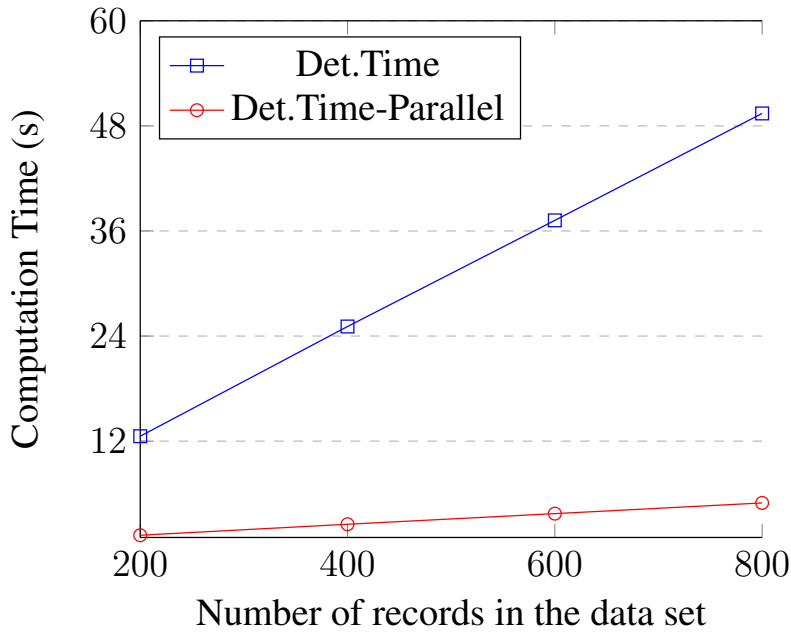
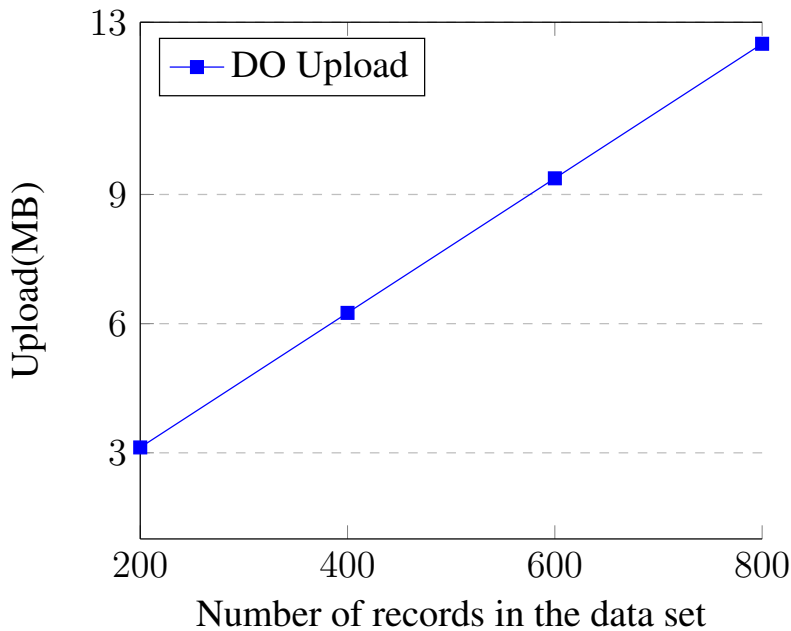


Figure 7.7: DO's upload depending on number of records in CIC-Friday-Afternoon with  $\ell = 512$ .



### 7.3.2 Performance Comparison of Three Classification Methods

In order to compare the performances of detection rules extracted from three classification methods (namely, binary decision tree (BDT), naïve Bayes and neural network (NN)) we conduct similar experiments on a machine with Intel core i7 running at 1.90 GHz processor and 32 GB of RAM. We adopt CIC IDS 2017 data sets [51], and kDD Cup 1999 [77]

Data set	Number of records	Number of records in the target (or attack) class	Number of features (d)	Max domain size of features (t)	Intrusion types
CIC-Friday-Morning	615	137	7	5	Bot
CIC-Friday-Afternoon	4062	2031	8	4	DDoS
KDD Cup 1999	1438	748	8	4	Smurf Neptune

Table 7.5: Data sets characteristics

data set from UCI repository [78] as benchmarks to report the detection performance of three rule-based classifiers. As explained previously, the CIC IDS 2017 data set captures network activity of five days in a test bed, designed by Canadian Institute for Cybersecurity (CIC) at University of New Brunswick. In the CIC IDS 2017 data set, we used random records from Friday morning and Friday afternoon such that the class distribution in original data set is preserved; data sets used here are not exactly same as ones in Section 7.3.1, because here we change the number of features and domain size of each feature as illustrated in Table 7.5. Friday morning data (CIC-Friday-Morning) includes network packets during Botnet attacks while Friday afternoon data (CIC-Friday-Afternoon) is collected during DDoS attacks. We adopt the KDD Cup 1999 data set since it is the most widely used data set for the evaluation of intrusion detection systems.

Table 7.5 illustrates the specifications regarding to each data set. The first column of the table represent the name of the data sets; the second column of the table shows the number of records in the data set. The third column represents the number of records which are labelled as intrusion in each data set. The fourth and fifth columns show the number of features, and the maximum domain size of features regarding to each data set, respectively. The last column in the table shows intrusion types existing in each data set. In Figure 7.8, we compare the detection performance of three rule-based classifiers, provided in Chapter 4, on CIC-Friday-Morning data set. The result shows that, all three provided classifiers yield high performance for this data set. The closer the precision and recall rate are to 1, the better the classification.

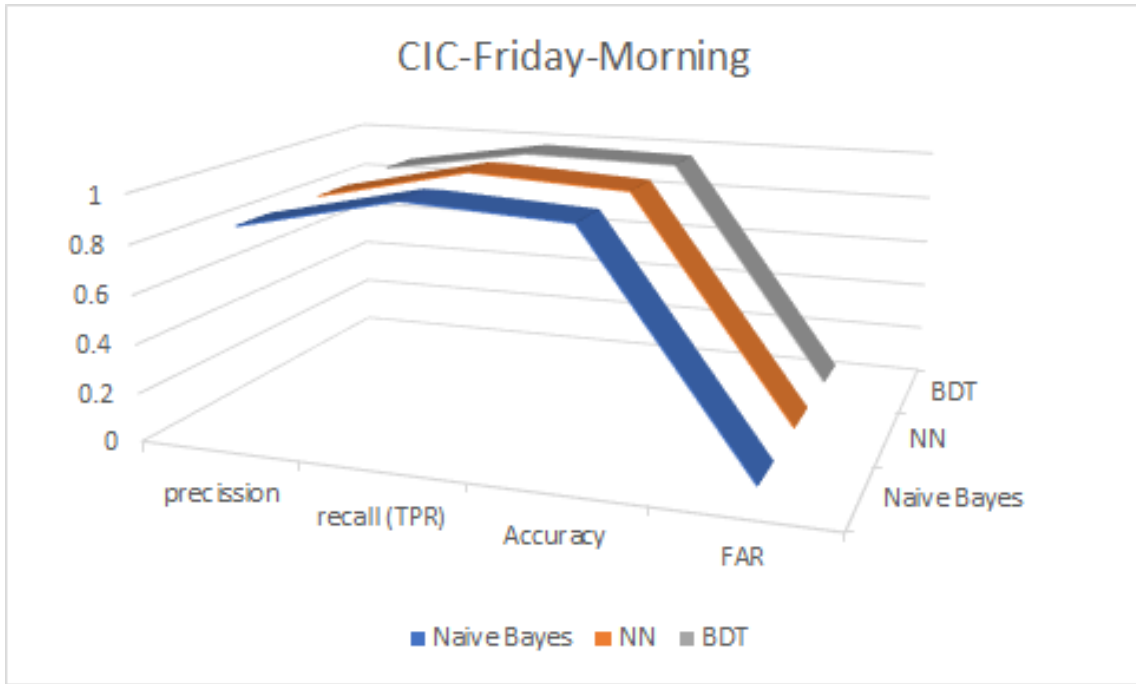


Figure 7.8: Comparison of 3 rule-based classifier on samples of the CIC-Friday-Morning data set.

In Figure 7.9, we compare the detection performance of three rule-based classifiers, on random samples from the CIC-Friday-Afternoon data set. The results show that the decision tree and neural network classifiers perform almost same; however the naïve Bayes classifier has worse detection rate (recall), while its accuracy is higher in comparison with the two other classifiers.

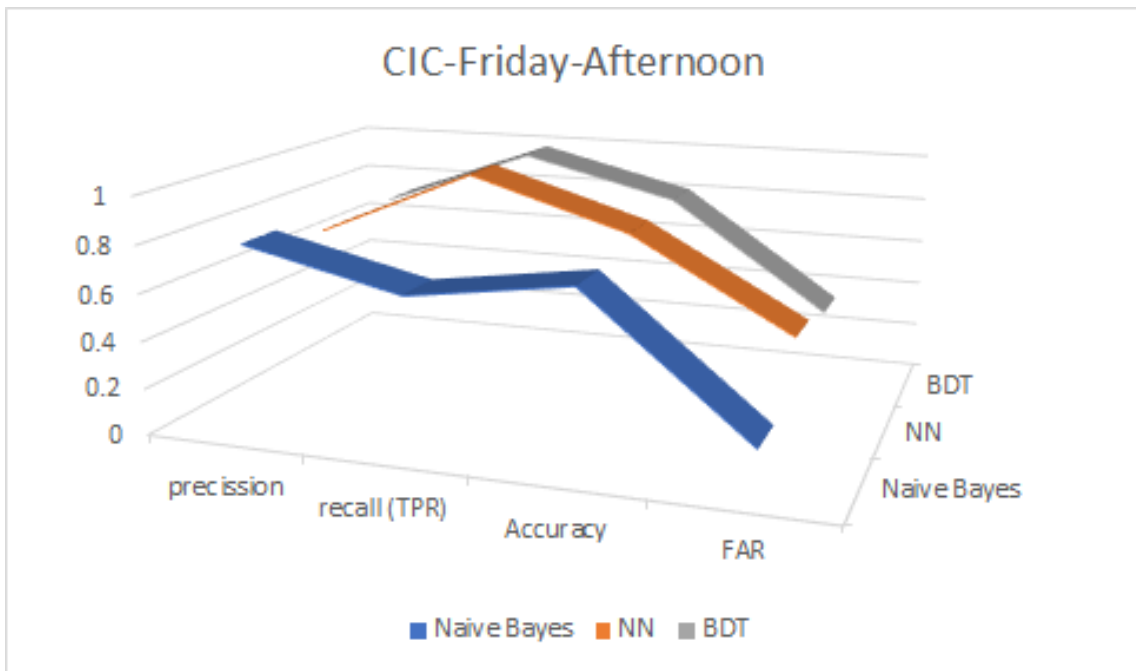


Figure 7.9: Comparison of 3 rule-based classifier on sample of the CIC-Friday-Afternoon data set

In Figure 7.10, we compare the detection performance of three rule-based classifiers, on samples of the KDD Cup 1999 data set. The results show that the precision rates of the naïve Bayes and the NN classifiers are better than the BDT classifier. The recall rate of the NN classifier is worse than the two other methods. In term of accuracy rate, the naïve Bayes and BDT classifiers perform almost the same. FAR of the BDT classifier is higher than the two other classifiers.

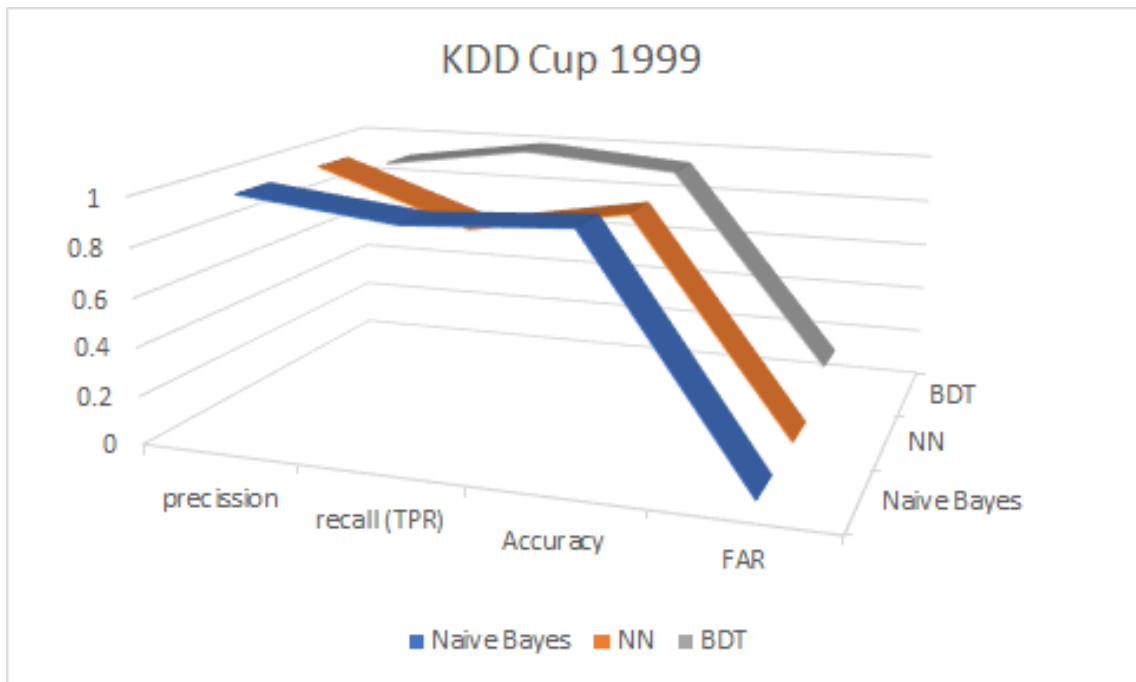


Figure 7.10: Comparison of 3 rule-based classifier on sample of the KDD Cup 1999 data set

Table 7.6 shows the number of attack rules extracted from each rule-based classifier. The number of attack rules extracted from the naïve Naves classifier is much higher in comparison with the two other classifiers. The reason is due to the fact that attack rules are extracted from the space of all possible combinations of values of features, whose size is exponential in the number of features. Thus, if the number of features is too large, the naïve Bayes classifier can become impractical in extracting rules. However, we find out that not all the rules, which are extracted by the naïve Bayes classifier, are used in detecting attacks in the test data set. Thus, by selecting a reasonable threshold, we can decrease the number of attack rules extracted by the naïve Bayes classifier significantly.



Data set	BDT	Naïve Bayes	NN
CIC-Friday-Morning	1	54	5
CIC-Friday-Afternoon	2	106	9
KDD Cup 1999	2	57	1

Table 7.6: Number of attack rules extracted by each rule-based classifier

Table 7.7 shows the computation times of all three classifier to extract rules from the three data sets. The BDT classifier is the fastest whereas the NN classifier is the slowest of the three.

Data set	BDT	Naïve Bayes	NN
CIC-Friday-Morning	46	7,893	12,733
CIC-Friday-Afternoon	80	11,983	221,912
KDD Cup 1999	74	13,300	47,218

Table 7.7: Rule extraction time (millisecond)

Table 7.8 demonstrates the detection time of encrypted attack rules on DO data. The detection time depends on number of records, number of attack rules, and signature size. Among the three, the naïve Bayes classifier yields the slowest detection time due to the large number of attack rules.

Data set	BDT	Naïve Bayes	NN
CIC-Friday-Morning	36	2,123	180
CIC-Friday-Afternoon	218	60,200	2,205
KDD Cup 1999	170	5,181	87

Table 7.8: Intrusion detection time (millisecond)

As a conclusion, we can safely claim that the BDT classifier always outperform in terms of execution times due to the fewer number of attack rules to a large extent. Thus, the BDT classifier is more efficient in rule selection. Also, its precision, recall, accuracy and FAR values are always acceptable and compares favorably with the two other classifiers most of the time. Notwithstanding, some performance figures of the two other classifiers are better than those of BDT for some data sets. This result suggests that we can profitably use different rule extraction methods and employ the best rules together to improve the overall performance of the intrusion detection scheme.

# Chapter 8

## Conclusion

In this work, we presented a highly efficient and effective protocol for intrusion detection over network data where both the detection model and network data are private. We used primarily a decision tree as an intrusion policy that is privately evaluated over network data using homomorphic encryption algorithms. Compared to other private evaluation protocols of decision trees in the literature, the proposed protocol is novel in the sense that it is tailored to meet the privacy requirements of intrusion detection applications and incorporate several optimization techniques for efficiency and effectiveness. We provided a set of security definitions and showed that the proposed protocol is secure against semi-honest adversaries. For malicious adversaries, we sketch most feasible attack methods and evaluate their feasibility. Our analyses demonstrate that security and privacy can be enhanced by adjusting several parameters such as number of rules, number of features, the number of categories in discretization of features. We also show that the proposed discretization method proves to be useful not only in increasing the efficiency in terms of execution time but also in enhancing the privacy.

For the first time in the literature we formally addressed the effect of a priori information given to data owner such as features and their domains and the predicates at decision nodes, on the privacy of the detection model.

We implemented the protocol and evaluated its performance on decision trees trained on several real-world data sets. Our experiments demonstrate that our protocol compares favorably with existing protocols in the literature in terms of both computation and communication. The execution times indicate that the protocol can be used in real life applications as the protocol is fast, scalable and amenable to parallel computation.

We also explored other methods to extract attack rules such as naïve Bayes and neural networks and compare their performances with the binary decision tree classifier. Even though binary decision tree based classifier is generally faster in extracting attack rules and usually performs better as far as correctness is concerned than the two other methods, we observed that for certain correctness metrics the former can be outperformed by the latter methods for some data sets. This observation motivates the investigation of other

rule extraction techniques and usage of rules from different extraction techniques together to improve the overall performance of the intrusion detection system.

In this dissertation, we show that our protocol can be used in any rule-based intrusion detection scheme independent of the methods used to extract detection rules. As future work, we aim to employ intrusion rules extracted from different methods, which can give different and perhaps conflicting classification decisions for the same record. Therefore, a more involved technique to reconcile the incompatible decisions is needed. This, in return, necessitates modifications to our basic protocol as well as to security and privacy definitions. Also, we plan to extend the experiments using larger data sets and more features.

# Bibliography

- [1] S. Singh and S. Silakari, “A survey of cyber attack detection systems,” *International Journal of Computer Science and Network Security*, vol. 9, no. 5, pp. 1–10, 2009.
- [2] H.-A. Park, D. H. Lee, J. Lim, and S. H. Cho, “Ppids: privacy preserving intrusion detection system,” in *Pacific-Asia Workshop on Intelligence and Security Informatics*. Chengdu, China: Springer, Berlin, 11-12 April 2007, pp. 269–274.
- [3] S. Niksefat, P. Kaghazgaran, and B. Sadeghiyan, “Privacy issues in intrusion detection systems: A taxonomy, survey and future directions,” *Computer Science Review*, vol. 25, pp. 69–78, 2017.
- [4] M. V. Mahoney and P. K. Chan, “An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection,” in *International Workshop on Recent Advances in Intrusion Detection*. Pittsburgh, PA, USA: Springer, Berlin, 8-10 September 2003, pp. 220–237.
- [5] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.
- [6] S. Micali and P. Rogaway, “Secure computation (abstract),” in *CRYPTO ’91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*. Santa Barbara, California, USA: Springer-Verlag, UK, 11-15 August 1991, pp. 392–404.
- [7] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford: Stanford University, 2009.
- [8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.
- [9] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

- [10] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.
- [11] A. Bivens, C. Palagiri, R. Smith, B. Szymanski, M. Embrechts *et al.*, "Network-based intrusion detection using neural networks," *Intelligent Engineering Systems through Artificial Neural Networks*, vol. 12, no. 1, pp. 579–584, 2002.
- [12] M. Amini, R. Jalili, and H. R. Shahriari, "Rt-unnid: A practical solution to real-time network-based intrusion detection using unsupervised neural networks," *Computers & Security*, vol. 25, no. 6, pp. 459–468, 2006.
- [13] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Technical report, Tech. Rep., 2000.
- [14] A. O. Adetunmbi, S. O. Falaki, O. S. Adewale, and B. K. Alese, "Network intrusion detection based on rough set and k-nearest neighbour," *International Journal of Computing and ICT Research*, vol. 2, no. 1, pp. 60–66, 2008.
- [15] H. Zhengbing, L. Zhitang, and W. Junqi, "A novel network intrusion detection system (nids) based on signatures search of data mining," in *Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop*. Adelaide, Australia: ICST, Belgium, 21 - 23 January 2008, pp. 45:1–45:7. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1363217.1363274>
- [16] G. Agrawal, S. K. Soni, and C. Agrawal, "A survey on attacks and approaches of intrusion detection systems," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 8, pp. 499–504, 2017.
- [17] B. Lokesak, "A comparison between signature based and anomaly based intrusion detection systems," *PPT*). [www.iup.edu](http://www.iup.edu).
- [18] V. Kumar and O. P. Sangwan, "Signature based intrusion detection system using snort," *International Journal of Computer Applications & Information Technology*, vol. 1, no. 3, pp. 35–41, 2012.
- [19] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM, 2003, p. 25–36.
- [20] F. M. Groom, K. Groom, and S. S. Jones, *Network and Data Security for Non-engineers*. Auerbach Publications, 2016.
- [21] K. Thomas, "Sony makes it official: Playstation network hacked," *PC Computing*, p. 12, 2011.

- [22] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, “A guide to fully homomorphic encryption,” Cryptology ePrint Archive, Report 2015/1192, 2015, <https://eprint.iacr.org/2015/1192>.
- [23] L. Morris, “Analysis of partially and fully homomorphic encryption,” *Rochester Institute of Technology*, pp. 1–5, 2013.
- [24] M. Belland, W. Xue, M. Kurdi, and W. Chu, “Somewhat homomorphic encryption,” 2017.
- [25] X. Yi, R. Paulet, and E. Bertino, *Fully Homomorphic Encryption*. Cham: Springer International Publishing, 2014, pp. 47–66. [Online]. Available: [https://doi.org/10.1007/978-3-319-12229-8\\_3](https://doi.org/10.1007/978-3-319-12229-8_3)
- [26] M. Mohan, M. K. Devi, and V. J. Prakash, “Homomorphic encryption-state of the art,” in *2017 International Conference on Intelligent Computing and Control (I2C2)*. IEEE, 2017, pp. 1–6.
- [27] D. Micciancio, “Lattice-based cryptography,” *Encyclopedia of Cryptography and Security*, pp. 713–715, 2011.
- [28] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” *Journal of the ACM (JACM)*, vol. 56, no. 6, p. 34, 2009.
- [29] ———, “Lattice-based cryptography,” in *Annual International Cryptology Conference*. Springer, 2006, pp. 131–141.
- [30] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.
- [31] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.
- [32] G. E. Hinton, T. J. Sejnowski, and T. A. Poggio, *Unsupervised learning: foundations of neural computation*. MIT press, 1999.
- [33] S. R. Safavian and D. Landgrebe, “A survey of decision tree classifier methodology,” *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.
- [34] L. A. Breslow and D. W. Aha, “Simplifying decision trees: A survey,” *The Knowledge Engineering Review*, vol. 12, no. 1, pp. 1–40, 1997.
- [35] K. M. Leung, “Naive bayesian classifier,” *Polytechnic University Department of Computer Science/Finance and Risk Engineering*, 2007.

- [36] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.
- [37] H. Lu, R. Setiono, and H. Liu, “Effective data mining using neural networks,” *IEEE transactions on knowledge and data engineering*, vol. 8, no. 6, pp. 957–961, 1996.
- [38] H. Viktor and I. Cloete, “Extracting dnf rules from artificial neural networks,” in *International Workshop on Artificial Neural Networks*. Springer, 1995, pp. 611–618.
- [39] S. V. Stehman, “Selecting and interpreting measures of thematic classification accuracy,” *Remote sensing of Environment*, vol. 62, no. 1, pp. 77–89, 1997.
- [40] D. M. Powers, “Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation,” 2011.
- [41] J. Brownlee, “Feature selection to improve accuracy and decrease training time,” *Machine Learning Mastery*, 2014.
- [42] G. Holmes, A. Donkin, and I. H. Witten, “Weka: A machine learning workbench,” in *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*. IEEE, 1994, pp. 357–361.
- [43] J. Benesty, J. Chen, Y. Huang, and I. Cohen, “Pearson correlation coefficient,” in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.
- [44] M. A. Hall, “Correlation-based feature selection for machine learning,” 1999.
- [45] A. G. Karegowda, A. Manjunath, and M. Jayaram, “Comparative study of attribute selection using gain ratio and correlation based feature selection,” *International Journal of Information Technology and Knowledge Management*, vol. 2, no. 2, pp. 271–277, 2010.
- [46] G. Chandrashekar and F. Sahin, “A survey on feature selection methods,” *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.
- [47] M. Lichman, “UCI machine learning repository,” 2013. [Online]. Available: <http://archive.ics.uci.edu/ml>
- [48] J. McHugh, “Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.



- [49] C. Brown, A. Cowperthwaite, A. Hijazi, and A. Somayaji, "Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhibit," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications. CISDA 2009*. Ottawa, ON, Canada: IEEE, New York, 8-10 July 2009, pp. 1–7.
- [50] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. ACM, 2011, pp. 29–36.
- [51] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSP*. Funchal, Madeira-Portugal: SCITEPRESS, 22-24 January 2018, pp. 108–116.
- [52] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.
- [53] N. B. Amor, S. Benferhat, and Z. Elouedi, "Naive bayes vs decision trees in intrusion detection systems," in *Proceedings of the 2004 ACM symposium on Applied computing*. Nicosia, Cyprus: ACM, New York, 14-17 March 2004, pp. 420–424.
- [54] L. Dias, J. Cerqueira, K. Assis, and R. Almeida, "Using artificial neural network in intrusion detection systems to computer networks," in *Computer Science and Electronic Engineering (CEECE)*. Colchester, UK: IEEE, New York, 27-29 September 2017, pp. 145–150.
- [55] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02*, vol. 2. Honolulu, HI, USA: IEEE, New York, 12-17 May 2002, pp. 1702–1707.
- [56] A. Alashqur, "A novel methodology for constructing rule-based naïve bayesian classifiers," *International Journal of Computer Science & Information Technology*, vol. 7, no. 1, p. 139, 2015.
- [57] B. Śnieżyński, "Converting a naive bayes models with multi-valued domains into sets of rules," vol. 4080, 09 2006, pp. 634–643.
- [58] B. Sniezynski, "Converting a naive bayes model into a set of rules," in *Intelligent Information Systems*, 2006.

- [59] H. K. Greenspan, R. Goodman, and R. Chellappa, “Combined neural network and rule-based framework for probabilistic pattern recognition and discovery,” in *Advances in Neural Information Processing Systems*, 1992, pp. 444–451.
- [60] C. M. Higgins and R. Goodman, “Incremental learning with rule-based neural networks,” 1991.
- [61] R. M. Goodman, C. M. Higgins, J. W. Miller, and P. Smyth, “Rule-based neural networks for classification and probability estimation,” *Neural computation*, vol. 4, no. 6, pp. 781–804, 1992.
- [62] R. Büschkes and D. Kesdogan, “Privacy enhanced intrusion detection,” *Multilateral security in communications, information security*, vol. 1999, pp. 187–204, 1999.
- [63] M. Sobirey, S. Fischer-Hübner, and K. Rannenber, “Pseudonymous audit for privacy enhanced intrusion detection,” in *Proceedings of the IFIP TC11 13 international conference on Information Security (SEC ’97)*. Copenhagen, Denmark: Springer, Boston, MA, 14-16 May 1997, pp. 151–163.
- [64] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, “Machine learning classification over encrypted data.” in *NDSS*. San Diego, California: Internet Society, 8-11 February 2015.
- [65] S. Halevi and V. Shoup, “Algorithms in helib,” in *International Cryptology Conference*. Santa Barbara, CA, USA: Springer, Berlin, 17-21 August 2014, pp. 554–571.
- [66] D. J. Wu, T. Feng, M. Naehrig, and K. Lauter, “Privately evaluating decision trees and random forests,” *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 335–355, 2016.
- [67] K. Bache and M. Lichman, “Uci machine learning repository,” 2013.
- [68] H. Chen, K. Laine, and R. Player, “Simple encrypted arithmetic library - seal v2.1,” in *International Conference on Financial Cryptography and Data Security*. Sliema, Malta: Springer, Berlin, 3-7 April 2017, pp. 3–18.
- [69] J. Huysmans, R. Setiono, B. Baesens, and J. Vanthienen, “Minerva: Sequential covering for rule extraction,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 2, pp. 299–309, 2008.
- [70] J. Hühn and E. Hüllermeier, “Furia: an algorithm for unordered fuzzy rule induction,” *Data Mining and Knowledge Discovery*, vol. 19, no. 3, pp. 293–319, 2009.

- [71] P. Smyth and R. M. Goodman, "An information theoretic approach to rule induction from databases," *IEEE transactions on Knowledge and data engineering*, vol. 4, no. 4, pp. 301–316, 1992.
- [72] N. Blachman, "The amount of information that y gives about x," *IEEE transactions on Information Theory*, vol. 14, no. 1, pp. 27–31, 1968.
- [73] R. M. Goodman and P. Smyth, "The induction of probabilistic rule sets—the itrule algorithm," in *Proceedings of the sixth international workshop on Machine learning*. Elsevier, 1989, pp. 129–132.
- [74] L. Karaçay, E. Savaş, and H. Alptekin, "Intrusion detection over encrypted network data," *The Computer Journal*, 2019. [Online]. Available: <https://doi.org/10.1093/comjnl/bxz111>
- [75] J. Novakovic, "Using information gain attribute evaluation to classify sonar targets," in *17th Telecommunications forum TELFOR*. Belgrade, Serbia: IEEE, New York, 24-26 November 2009, pp. 1351–1354.
- [76] S. R. Garner, "Weka: The waikato environment for knowledge analysis," *Proceedings of the New Zealand Computer Science Research Students Conference*, vol. 1995, pp. 57–64, 05 1995.
- [77] K. Cup, "Dataset," available at the following website <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, vol. 72, 1999.
- [78] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: <http://archive.ics.uci.edu/ml>