Framework for systematic design, analysis and evaluation of intrusion detection systems

Li, Zhuowei

2007

Li, Z. W. (2007). Framework for systematic design, analysis and evaluation of intrusion detection systems. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2481>

<https://doi.org/10.32657/10356/2481>

Nanyang Technological University

Downloaded on 26 Apr 2023 17:35:01 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**A FRAMEWORK FOR SYSTEMATIC DESIGN,
ANALYSIS AND EVALUATION OF INTRUSION
DETECTION SYSTEMS**

LI ZHUOWEI

**LI ZHUOWEI
SCHOOL OF COMPUTER ENGINEERING
2007**

A FRAMEWORK FOR SYSTEMATIC DESIGN, ANALYSIS
AND EVALUATION OF INTRUSION DETECTION SYSTEMS

2007

Leave it blank intentionally

A Framework for Systematic Design, Analysis and Evaluation of Intrusion Detection Systems

LI ZHUOWEI

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2007

Leave it blank intentionally

Acknowledgments

This thesis would not have been possible without the help of many people. First and foremost, I thank my supervisor, **Dr. Amitabha Das**. Not only did he provide the research orientation for me, he is always there to discuss with me, without any hesitation, no matter how strange my questions are. I would like to thank him for his patience and efforts to show me the way of doing a successful research, and also the frequent discussions of any ideas, extraordinary or traditional. I would like to thank him for his careful review of my writings, papers and reports, and his valuable suggestions. I will also thank **Dr. Jianying Zhou**, who acted as my co-supervisor during my attachment in I2R, A*Star. I would like to thank him not only for allowing me to participate in the research in I2R, but also for his instructions and suggestions to my research and my job hunting.

I am deeply indebted to the love and support of my family. First, I would like to express my deepest appreciation to my wife, Mrs. Yang Yang. Without her love, care, support and encouragement, I will not be able to continue my Ph.D. study. She is my greatest source of comfort and encouragement, and has endured as much anxieties as I have for this thesis. Many thanks are also due to my parents and my parents-in-law for their persistent support.

I would like to thank all CAISians for their kind help and suggestions, and also for making my life in CAIS immensely enjoyable, especially thanks to Zhao Qiankun and Weng Jianshu. Thanks also go to the School of Computer Engineering and Nanyang Technological University for the financial support and research facilities.

Contents

Acknowledgments	i
List of Figures	vi
List of Tables	viii
Abstract	x
1 Introduction	1
1.1 Intrusion Detection	2
1.2 Motivations	3
1.2.1 Generic Problems in Intrusion Detection	4
1.3 Our Contributions	6
1.4 Roadmap of the Dissertation	7
2 Literature Review	9
2.1 General Frameworks for Problem Analysis	10
2.2 Model Generalization	13
2.2.1 Extending Behavior Models	14
2.2.2 Eluding Intrusion Detection	15
2.3 Feature Selection	16
2.4 Behavior Signature Generation	18
2.4.1 Existing Intrusion Signatures	18
2.4.2 Worm Signatures	19
2.4.3 Signatures for Intrusion Variations	21
2.5 Suppressing False Alarms	21
2.6 Summary	22
3 A Systematic Methodology for Intrusion Detection	24
3.1 Background	24
3.2 A Comprehensive Framework	25
3.2.1 Basic Concepts and Notations	26
3.2.2 NSA Label	28

3.2.3	Compound Feature	29
3.2.4	An Illustrative Example	30
3.2.5	Behavior Signature	31
3.3	Intrusion Detection Problem	33
3.4	Venn Diagram Representation	34
3.5	Desirable Properties	35
3.6	Using the Framework	36
3.6.1	Its Applications	37
3.6.2	The Experimental Dataset	37
3.7	Summary	39
4	Problem Analysis	40
4.1	Preliminaries	40
4.2	Quantifying Behavior Spaces/Models	41
4.2.1	Model Inaccuracy	42
4.2.2	Model Incompleteness	42
4.3	Problem Analysis	43
4.3.1	Signature-based Intrusion Detection	44
4.3.2	Anomaly-based Intrusion Detection	46
4.3.3	USAID: Unifying SID and AID	49
4.4	Further Discussions	50
4.5	Summary	51
5	Model Generalization and Its Implications on Intrusion Detection	52
5.1	An Evaluation Framework	53
5.1.1	L0 Without Generalization	55
5.1.2	L1 Model Generalization	55
5.1.3	L2 Model Generalization	57
5.1.4	L3 Model Generalization	61
5.1.5	Measuring the Detection Performance	61
5.2	Experiments: A Case Study	62
5.2.1	Without Model Generalization	62
5.2.2	Evaluating L1 Model Generalization	63
5.2.3	Evaluating L2 Model Generalization	64
5.2.4	Evaluating L3 Model Generalization	66
5.2.5	The Implications of Model Generalization	67
5.3	Summary	68

6	Unifying Signature-based and Anomaly-based Intrusion Detection	69
6.1	Feature Subspaces Extraction	70
6.2	Building the NSA Signature Base	71
6.3	A Detection Mechanism	71
6.4	Experimental Results on USAID	72
6.4.1	Feature Ranges of Every Feature	73
6.4.2	The NSA Signature Base	73
6.4.3	Detection Results	75
6.4.4	Performance Comparisons	77
6.5	Summary	78
7	varUSAID: USAID with Variable Length Signatures	80
7.1	Variable-length Signatures	80
7.2	USAID with Variable-length Signatures	81
7.2.1	Feature Subspaces Extraction	81
7.2.2	A Sorted Feature Vector	82
7.2.3	Building the NSA Signature Base	84
7.2.4	A Detection Mechanism via Variable-length Signatures	86
7.2.5	Mimicry Attacks on varUSAID	87
7.3	Experimental Results on varUSAID	87
7.3.1	Mining a Sorted Feature Vector	87
7.3.2	Building the NSA Signature Base	88
7.3.3	Detection Results	90
7.4	Summary	91
8	Automatic Signature Assemblers	93
8.1	Automatic Signature Assemblers	93
8.1.1	Discovering Feature Ranges for Assemblers	94
8.2	Experiments: A Case Study	97
8.2.1	Evaluating S0~S3 Splitting Strategies	97
8.3	Summary	100
9	Anomaly Context Identification	102
9.1	Anomaly Context Identification	103
9.1.1	Anomaly Contexts of an Intrusion	104
9.1.2	Effectiveness of ACI	105
9.1.3	An IDS Architecture with ACI	106

9.1.4	Other uses of the ACI scheme	110
9.2	Anomaly Contexts in stide: A Case Study	110
9.2.1	Notations and Definitions	110
9.2.2	Anomaly Contexts in stide	111
9.2.3	Mining Anomaly Contexts from Audit Trails	112
9.3	Evaluations	114
9.3.1	Visualizing Anomaly Contexts	115
9.3.2	Identifying Minimal Anomaly Contexts	117
9.4	Further Discussions	120
9.4.1	Put it Into Our Box: ACI with USAID	120
9.5	Summary	121
10	Conclusions and Outlook	123
10.1	Summary	123
10.2	Thesis Contributions	125
10.3	Future Work	127
10.4	Concluding Remarks	129
	References	130
A		140
A.1	The Sorted Feature Vector	140
A.2	Acronyms	140
B	List of Publications	142

List of Figures

1.1	Software development in ideal world (a) and real world (b).	2
3.1	The Venn diagram of the audit trail instance sets. $I_n \subseteq I_u$ and $I_a \subseteq I_u$	35
4.1	Behavior spaces and models for intrusion detection.	41
4.2	The intrusive behavior model $A_r(F)$ in SID techniques, where (1) $e+f+g = 1$, (2) $ S_i(F) = x * N_i(F) $, $ A_i(F) = y * N_i(F) $, (3) $ AS_1(F) + AS_2(F) =$ $f * A_r(F) $, (4) $ AN_1(F) = g * A_r(F) $	44
4.3	The normal behavior model $N_r(F)$ in AID techniques, where (1) $a+b+c = 1$, (2) $ S_i(F) = x * N_i(F) $, $ A_i(F) = y * N_i(F) $, (3) $ NS_1(F) + NS_2(F) =$ $b * N_r(F) $, (4) $ NA_1(F) = c * N_r(F) $	46
5.1	L2: model generalization - grouping (nMutate=0).	65
5.2	L2: model generalization - mutation (nGroup=3).	66
6.1	A general architecture for USAID.	69
6.2	Detection results of USAID (%).	75
7.1	A general architecture for varUSAID.	81
7.2	Four scenarios of the compounding between $F_{1...i}$ and F_{i+1} . The increase of the identification capacity is due to the compounding operations represented by red bold lines between F_{i+1} and $F_{1...i+1}$	82
7.3	Building the NSA signature base in varUSAID.	85
7.4	Percentage of instances identified by the increasing FV_{sort}	88
8.1	The workflow of automatic signature assemblers.	93
8.2	Splitting strategies for an unknown feature subspace.	97
9.1	A novel architecture of an intrusion detection system.	106
9.2	The anomaly context identification module.	107
9.3	The foreign sequence graphs for process ‘named’ and intrusion ‘buffer overflow’.	116

9.4	Foreign sequence graphs. The intrusions are: (a) ‘buffer overflow’; (b) ‘lprcp’; (c) ‘syslog’; (d) ‘sm565a’ and ‘sm5x’; (e) ‘decode’; (f) ‘forward loops’; (g) ‘sunsendmailcp’; (h) ‘misconfiguration’; (i) ‘buffer overflow’.	117
9.5	The number of MFS sequences (non-duplicated) in all intrusive datasets. . .	118
9.6	An integral intrusion detection system with three functional modules. . . .	121

List of Tables

3.1	The notations in our framework.	25
3.2	The example instances of the feature vector. We use ‘I’ to denote the status ‘intrusion’, and ‘Normal’ to represent normal behaviors.	30
3.3	The feature vector from connection records.	38
3.4	Intrusions in the training and test datasets.	38
4.1	Detection results v.s. true classes. The rows correspond to the detection outcomes, and the columns correspond to the true classes of resource behaviors in question.	43
5.1	The behavior signatures built from the instances in Table 3.2, where ‘I’ denotes the status ‘intrusion’, and ‘Normal’ represents normal behaviors. We have used the L1.2 splitting strategy and the range merging step in building feature ranges. We only list signatures here to illustrate L2 and L3 model generalization.	58
5.2	Detection results and their costs.	62
5.3	L0: without model generalization. The 2nd and 3rd columns give the numbers of normal and intrusive instances in every sample pair respectively. . .	62
5.4	L1: model generalization (L1.1~4)	63
5.5	L3: model generalization (nGroup=3, nMutate=5)	67
5.6	The significance of different levels of model generalization. The symbol ‘↓’ represents ‘ decrease ’ and the symbol ‘↑’ represents ‘ increase ’. ‘-’ denotes that it will not affect the parameter. TP: true positive ($\#_{II} + \#_{IA}$), FP: false positive ($\#_{NA}$), TN: true negative ($\#_{NN}$), FN: false negative ($\#_{IN}$). .	68
6.1	Feature ranges of the discrete feature “hot”.	71
6.2	The anomaly classes in USAID. In a class name, ‘ D ’ represents that it is in the detection results, ‘ -I ’ represents that it is identified as an intrusion, ‘ -S ’ represents that it is identified by a suspicious signature, ‘ -A ’ represents that it is an anomaly.	72

6.3	The statistics of shared signatures.	75
6.4	False alarm rates w.r.t. anomaly categories.	77
6.5	The detection results of USAID. The horizontal dimension is the predicted class of every test example, and the vertical dimension is its actual class. The first 5 columns constitute the confusion matrix of USAID, and the last column includes the total number of anomalies for all anomaly classes except D-I.	78
7.1	Several examples of variable-length signatures.	85
7.2	The constitutions of NSA signature base in USAID and varUSAID.	88
7.3	Number of signature variations in the training audit trails.	89
7.4	Performance parameters in our experiments.	90
8.1	Notations used in detection results.	98
8.2	S0: without label assigning and splitting.	98
8.3	Evaluating S1, S2 and S3 splitting strategies.	99
8.4	Evaluating S1, S2 and S3 splitting strategies on the whole dataset.	100
9.1	The dataset specifications.	115
9.2	Shared MFS sequences of the same intrusion into the same process.	119
A.1	The features in the sorted feature vector FV_{sort}	140
A.2	A list of acronyms.	141

Abstract

Intrusion detection has become a critical component of the security infrastructure in the face of increasing vulnerabilities and threats in Internet applications. However, most intrusion detection techniques are stymied by a number of fundamental and practical problems, such as intrusion variation detection, higher false alarm rate. In this thesis, we will refer to these problems as the generic problems of intrusion detection, and propose a systematic approach to identify the root cause of these problems, analyze them and devise several solutions to address them.

We first introduce a comprehensive methodology for intrusion detection, where it mimicks the human behaviors in object identification by replacing precise feature values in an instance with value ranges. Within the framework, model generalization is analyzed and identified as the root cause of most generic problems, and thus evaluated experimentally to study its significance and impact for intrusion detection.

Using the analysis and evaluation results, we then devise a novel technique, which differs from existing techniques that only utilize partial knowledge of a resource, while our technique uses all available knowledge for intrusion detection. It performs better in detecting known and unknown intrusions with a lower false alarm rate. The only disadvantage of this technique is its intensive computation, but our remedial technique has made up for it successfully with negligible performance penalty.

We also propose two solutions to the generic problems. A signature assembly line is designed and validated to generate behavior signatures automatically. We proposed and validated the anomaly context identification scheme, in which the anomaly contexts of every alarm are used to make up for knowledge deficiency in intrusion detection. The false alarm rate can be lowered using the anomaly contexts of alarms.

Chapter 1

Introduction

The concern regarding information security has moved to the center stage since the emergence of the Internet. The Internet has brought about great benefits to the modern society, but its complexity, accessibility, and openness have served to increase the security risks of computing resources tremendously. According to the survey conducted by CSI/FBI [Pow02], the Internet has become a more frequent attack point. More and more skilled attackers emerge as the main sources of attacks, such as the notorious *Nimda*, *MSBlast*. The total annual losses related to computer and network systems has increased dramatically year by year due to the security vulnerabilities within most computing applications.

The vulnerabilities are ubiquitous in the internetworking applications because of the flawed software development in the real world (Figure 1.1 [SHF97]). By exploiting these vulnerabilities, an attacker can get what he wants from anywhere for any purpose, and also misuse or spoil the network infrastructures (e.g., *denial of services*). It is both technically difficult and expensive to ensure that a computing resource will not be harmed if these vulnerabilities are exploited with malicious intentions. So there are urgent requirements and reasons for security techniques by which the occurrence of the attacks can be stopped, detected and/or prevented.

Nowadays, there exist two-layer solutions addressing information security. One layer is a **proactive security solution**, such as user authentication (e.g. *using password and biometrics*), avoiding programming errors, and information protection (e.g., *encryption*), and it has been used as the first line of defense. However, security on a network goes way beyond encryption of data, and the solution itself cannot guarantee the complete security of a system. The second is a **reactive security solution**,

Figure 1.1: Software development in ideal world (a) and real world (b).

such as firewall, Intrusion Detection System or IDS [And80]. The two solutions are complementary, any of them could not build a secure system alone. As the last line of defense, intrusion detection techniques should be given enough attention to build a useful and reliable security infrastructure.

1.1 Intrusion Detection

Historically, intrusion detection is first formally introduced by the seminal work of Anderson [And80] in 1980. After that, Denning's 1987 paper [Den87] has been highly influential for the later research on intrusion detection, and it has set up the basic principles for intrusion detection. Since then, many intrusion detection techniques are proposed, and many intrusion detection systems are developed, such as Haystack [Sma88], NSM [MHL94], EMERALD [PN97], MADAM ID [LS00], LERAD [MC02b]. Even though most questions raised by Denning [Den87] are still left unanswered [McH01], there are several useful and effective products in the market at present, such as Snort [Roe99], RealSecure [Rea].

In principle, intrusion detection is a process of intelligently monitoring the events occurring in a computing resource, the purpose of which is to analyze these events for

successful and unsuccessful attempts to abuse the computing resources. To achieve the purpose, it is always assumed that an intrusion will manifest itself in a trace of these events, and the trace of an intrusion is different from other traces left by normal behaviors. In other words, only the intrusions that meet the above requirements are detected. From the perspective of implementation, an IDS system usually consists of three components: (1)the *information sources* that provide the stream of audit trails; (2)the *analysis engines* that find manifestations of intrusions; and (3)the *response components* that generate reactions if an intrusion is found.

The information source component captures events from the audit trails of a resource and provides them to the event analysis engines. The events may be a sequence of commands executed by a user, a sequence of system calls launched by an application (e.g., sendmail), a sequence of network packets, and so on. According to the source of events, an IDS system can be further classified into two categories: *host-based* and *network-based*. A host-based IDS tries to identify intrusions by analyzing activities at hosts, mainly users and processes; while a network-based IDS focuses on analyzing the traffic that is transported over the network [MHL94] (e.g. Snort [Roe99] and Bro [Pax99]). Many successful techniques and tools have been developed to capture the events from these data sources, such as libpcap for Linux [Lib], Solaris BSM [HLF⁺01], strace for Linux [FHSL96], winpcap for windows [Win].

In the event analysis engines, based on the predefined behavior models, the event sequences are analyzed to indicate the possible anomalies or intrusions in the usage of the resource. Its input is the event sequence of a resource (*online or offline*), and its output is the decision whether current running of the resource is anomalous. Whenever an intrusion is identified, the third component is triggered to eliminate attack cause/effect. Recently, along with the response mechanisms, a new term has been coined to represent IDS, ‘Intrusion Prevention Systems’ [Des03]. However, if the central analysis engines are kept unchanged, ‘intrusion prevention’ will only be a hoax of market strategies. In this thesis, we focus on the analysis engines.

1.2 Motivations

In general, there exist two approaches for intrusion detection in the analysis engines: *signature-based intrusion detection* (SID, a.k.a., misuse detection), and *anomaly-based*

CHAPTER 1. INTRODUCTION

intrusion detection (AID). In SID, the intrusion signatures must be known beforehand and described a priori. An intrusion is detected if its behavior matches existing intrusion signatures. In comparison, a normal behavior model of a computing resource must be known beforehand in AID, and an intrusion to the computing resource is detected if the resource behavior deviates from the normal behavior model significantly.

From the perspective of signal detection, SID only uses the knowledge of *signals* (i.e., *intrusions*), whereas AID only uses the knowledge of *noise* (i.e., *normal behaviors*) for intrusion detection. In comparison, the classical signal detectors use the information of both signal and noise to make a decision. This information is about the distributions of signals or noise, which are known in advance in classical signal detectors. Correspondingly, one of the main tasks in intrusion detection is to characterize the distributions of signals (i.e., intrusions) and/or noise (i.e., normal behaviors). However, the Internet and computing applications, that are monitored by intrusion detection systems, are so diverse that the diversity increases the difficulty in this distribution characterization. For this reason, both approaches suffer from their respective fundamental and practical problems. In this thesis, we will refer to these problems as the generic problems of intrusion detection.

1.2.1 Generic Problems in Intrusion Detection

Each intrusion detection approach has its own strengths and weaknesses. SID can identify known intrusions *but cannot detect novel intrusions* (which is denoted as the problem $\mathcal{P}_S^1$), and AID can detect all intrusions *but cannot identify any intrusion from the audit trails* (denoted as $\mathcal{P}_A^1$). $\mathcal{P}_S^1$ and $\mathcal{P}_A^1$ are the respective fundamental problems of the two approaches, which cannot be solved using the corresponding approach (i.e., SID or AID) alone. Besides the fundamental problems, both approaches have their own practical ones respectively.

1.2.1.1 Signature-based Intrusion Detection

SID has the following practical problems, which limit its practical usefulness.

$\mathcal{P}_S^1$: **incapable of detecting new intrusions.**

$\mathcal{P}_S^2$: **signature updating bottleneck.**

The detection performance of SID is completely dependent on the intrusion

CHAPTER 1. INTRODUCTION

signature base. It is necessary to update the signature base instantly after a new intrusion occurs. However, the manual signature construction and the various signature updating constraints impede its signature deployment. Moreover, the fast-spreading (*e.g.*, *flash worms* [SMPW04]) intrusions make the problem more significant as a bottleneck in SID.

$\mathcal{P}_S^3$: **intrusion variation detection.** (Rubin et al.[6])

Different instances of an intrusion may cause different audit trails when the intrusion runs on a computing resource. These instances are called intrusion variations to each other. However, most intrusion signatures are extracted from a specific instance, and thus, if designed improperly, the intrusion signature cannot detect the other intrusion instances. Essentially, this is a signature completeness problem. This problem is aggravated by manual and automatic intrusion variation generators (Whisker [Whi99], Rubin et al. [RJM04], Vigna et al. [VRB04]).

$\mathcal{P}_S^4$: **too many false alarms.** (Julisch [Jul03])

As mentioned before, the intrusion signatures are manually constructed by security experts and it is difficult to guarantee the signature quality. This lack of quality control is manifested in the fact that some intrusion signatures even cover normal behaviors. Thus, such signatures will cause false alarms in the detection phase. At present, most existing IDS products have too many false alarms that reduce their usefulness in real deployment [Jul03].

1.2.1.2 Anomaly-based Intrusion Detection

AID also suffers from its own practical problems,

$\mathcal{P}_A^1$: **inability in identifying intrusions.**

$\mathcal{P}_A^2$: **higher false alarm rate.** (Axelsson [Axe99])

Under most scenarios, the normal behavior model is always incomplete because it is trained from existing audit trails. Every behavior outside the normal behavior model triggers an alarm in AID. Therefore, the outside normal behaviors cause false alarms. Same as in SID, too many false alarms will make AID useless due to the base rate fallacy [Axe99].

$\mathcal{P}_A^3$: **difficulty in identifying anomalies caused by intrusions.** (Li et al. [LD04b])

In AID, anomalies may be caused by some failures other than intrusions. Due to the lack of intrusion identification, it is difficult to determine whether an

anomaly is caused by an intrusion. This problem will be further aggravated since normal behaviors outside incomplete normal behavior models will lead to anomalies as well.

$\mathcal{P}_A^4$: **mimicry attacks.** (Wagner et al. [WS02])

In order to cover more normal behaviors, the normal behavior model is always generalized from the training audit trails. However, the generalization procedure is two-edged. The normal behavior models will cover some intrusive behaviors as well, which are exploited by mimicry attacks to avoid being detected.

$\mathcal{P}_A^5$: **concept drifting problem.**(Li et al. [LD04a])

With the change of time, the normal behavior models may be evolved. It will make the above three problems more serious.

$\mathcal{P}_A^6$: **intensive detection computation.**

Most importantly, the computational cost of intrusion detection must be reduced considerably before it can be usefully deployed in practical systems. This is more so as the size of audit trails keep on growing.

In order to avoid or alleviate the abovementioned problems, the current research trend on intrusion detection has largely been the development of ad-hoc intrusion detection techniques for different applications (e.g., worm detection [GSQ⁺04, SEVS04], static analysis [WD01]). However, the advancement of Internet and computing applications increases the number of application scenarios for intrusion detection. This makes it impractical to develop ad-hoc techniques to handle all these application scenarios.

1.3 Our Contributions

Our objective is to find an integral solution or a set of solutions to solve or alleviate most generic problems for intrusion detection. To achieve it, we try to analyze these generic problems to identify their root causes, and evaluate the root causes for their implications on intrusion detection. So far, we have made the following contributions.

- Mimicking the object identification process in human beings, it proposes a formal framework for intrusion detection. Under the framework, the following systematic work is performed.
 - It analyzes the generic problems in intrusion detection *theoretically* in a quantified manner, and their root causes are identified.

CHAPTER 1. INTRODUCTION

- It evaluates the root causes *experimentally* using existing audit trails to figure out their implications on intrusion detection.
- Using the above analysis and evaluation results, it proposes a novel intrusion detection technique, called USAID. Like classical signal detectors, USAID uses all available knowledge about *signals* and *noise* to detect intrusions. The two problems due to knowledge shortage, $\mathcal{P}_S^1$ and $\mathcal{P}_A^1$, are thus solved partially.
 - In addition, a mechanism is developed to reduce the computational overhead (cf. $\mathcal{P}_A^6$) in detection by ranking features utilized and shortening the signatures. We also quantify the influence of the reduced computation on the detection performance.
- Besides the above systematic work, it provides solutions for several generic problems. Although the two solutions are specific for some problems, other generic problems may be alleviated by them since all generic problems are correlated.
 - It proposes a signature assembly line to generate (*normal and intrusive*) signatures automatically using the proposed framework. Automatic signature generation is useful to alleviate two generic problems $\mathcal{P}_S^2$, $\mathcal{P}_A^5$ because the automatically generated signatures will speed up the signature updating greatly and keep up with the evolution of resource behaviors.
 - It proposes a practical scheme to suppress the false alarms using anomaly contexts of alarms. The anomaly contexts helps to cluster the alarms and make up for the knowledge deficiency, so it is useful to reduce the number of false alarms in the system ($\mathcal{P}_S^4$, $\mathcal{P}_A^2$).

While there are still several open problems for solutions, our proposed approach has the potential to solve them in our future work (cf. Section 4.4).

1.4 Roadmap of the Dissertation

This thesis is organized in the following manner. Chapter 2 reviews the related literatures to our research. Chapter 3 provides a comprehensive framework for intrusion detection, Chapter 4 and 5 conducts theoretical and experimental analysis of generic problems. Guided by the analysis results, Chapter 6 proposes the novel technique USAID. To reduce the detection computation in USAID, Chapter 7 proposes varUSAID

CHAPTER 1. INTRODUCTION

using feature selection and variable-length signatures. Chapter 8 introduces the automatic signature assemblers to construct behavior signatures automatically. Anomaly context identification is described in Chapter 9 for the purpose of suppressing false alarms. In Chapter 10, we discuss new directions for our proposed framework, lay out the future work, and conclude the thesis.

Chapter 2

Literature Review

In the past 20 years, the research on intrusion detection has been conducted intensively, and a lot of techniques have been proposed. While all such techniques are largely related to our research, our focus in this chapter is to review literatures strongly related to our work. For details of existing techniques, we refer the reader to several comprehensive surveys (McHugh [McH01], Axelsson [Axe00a], Debar et al. [DDW00]). We will only take a look back the overall history and cornerstones on intrusion detection.

Historically, the concept of intrusion detection was introduced by Anderson [And80] in 1980, but its basic principles were first laid out formally by Denning in 1987 [Den87]. In the late 1980s, a number of notable systems (e.g., IDES [JV93], Haystack [Sma88], NADIR [HJS⁺93], Wisdom and Sense [VL89]) were developed for host-based audit trails, mostly relying on a combination of statistical and expert system approaches [McH01]. However, with the wide spread of internetworking applications, network security has become a main concern in providing information security. In this aspect, the Network Security Monitor [HDL⁺90] was well-known as an important cornerstone for intrusion detection because it led to the current trend of monitoring network traffic.

From the viewpoint of the underlying principles employed, the research on intrusion detection can be divided into two clear stages. At the first stage (1980 - earlier 1990s), the research emphasis was given to SID techniques, and thereafter several SID systems were developed and deployed for practical applications successfully, such as Snort [Roe99], RealSecure [Rea], Bro [Pax99]. However, the fast and widely spreading of intrusions (e.g., zero-day worms that exploit vulnerabilities even before they are known by security professionals) worsens the signature updating bottleneck, and they make SID inadequacy of detecting these instantly spreading intrusions. From the

CHAPTER 2. LITERATURE REVIEW

late 1990s, most research has turned to AID techniques, e.g., MADAM ID [LS00], LERAD [MC02b], PAYL [WS04]. While AID is more promising with the potential to detect novel intrusions, there are few successful products in the market till now.

As discussed in Chapter 1, most existing intrusion detection techniques are stuck with the various problems. The current research trend has largely been the development of ad-hoc intrusion detection techniques for different applications (e.g., worm detection [GSQ⁺04, SEVS04], program control flow [XDC04], static analysis [WD01]) mostly because some problems could be avoided or alleviated using such ad-hoc techniques under application-specific settings.

In comparison, our motivation is to develop a systematic approach to find reasonable and feasible solutions to these problems. To this end, we first investigate the existing literature to find how they have treated these problems. In particular, we concentrate on the work that analyzes the generic problems, and that proposes solutions to them. Section 2.1 reviews past frameworks for intrusion detection in which problem analysis is performed. Section 2.2 addresses model generalization, which is always manipulated to detect more novel behaviors or to elude detection in intrusion detection. As a potential method to reduce computation, feature selection is surveyed in Section 2.3. The existing literature on building behavior signatures and suppressing false alarms is reviewed in Section 2.4 and Section 2.5 respectively.

2.1 General Frameworks for Problem Analysis

There exist several frameworks of intrusion detection in the literature. We reviewed the efforts in applying the typical signal detection theory in intrusion detection (*pg. 11*), analyzing the tradeoffs between positive and negative detection schemes for anomaly detection (*pg. 12*), arguing the effectiveness of intrusion detection by the base-rate fallacy within Bayesian theory (*pg. 12-13*), analyzing detection rules to determine whether they meet the requirements of security policies and attack properties (*pg. 13*), and comparing existing techniques with respect to several performance parameters (*pg. 13-14*),

Signal Detection vs. Intrusion Detection. As a preliminary attempt [Axe00b], the traditional signal detection and estimation theory has been tried for the detection of intrusions. Following the nomenclature of signal detection, a *signal* in intrusion

CHAPTER 2. LITERATURE REVIEW

detection is an intrusive behavior, and the *noise* is a normal behavior. The comparisons between classical signal detection and intrusion detection were performed with respect to the source, the probabilistic transition mechanism, the observation space and the decision rule. In the source, the difference is that there are multiple signals (e.g., intrusions) for intrusion detection in comparison with one signal in typical detection theory. There is another notable difference in their respective observable audit trails. The audit trails for intrusion detection are discrete but in the traditional signal detection the input data is continuous. Through the discussions on the principles of anomaly detection and signature detection within the context of signal detection, the author has argued that the knowledge of both distributions of normal and intrusive behaviors will greatly enhance the performance of intrusion detection.

The analysis makes us believe that it is hard to apply the traditional signal detection techniques into intrusion detection directly. The problem domain is different: there is **continuous** data for **one signal** in signal detection but there are **discrete** audit trails for **multiple signals** in intrusion detection. Moreover, it is difficult to get the distributions of normal and intrusive behaviors in the dynamic and diverse environment for intrusion detection. In comparison, our framework tackles the exact problem on intrusion detection without considering the behavior distributions. Significantly, the importance of our research effort in this thesis has been emphasized by the following viewpoint in the same paper [Axe00b] that “*the researchers seldom present any empirical data or theoretical calculations of the required and expected performance of their prototype - either in terms of detection accuracy, false alarm rate, or runtime performance - instead limiting their arguments to purely architectural issues of the intrusion detection system.*”

Positive Detection vs. Negative Detection. In [EFH02], a formal framework was proposed to analyze the tradeoffs between positive and negative detection schemes in anomaly detection. The resource behaviors are represented by l -length binary strings in the framework, i.e., the activity patterns of a resource. The anomalousness of the next resource behavior is dependent only on a sample of existing behaviors (i.e., the current state of knowledge) of a legitimate process. According to a generation rule (e.g., r -contiguous bits, r -chunks, or hamming distance), the current sample can be generalized to cover more behaviors, which are not in the original sample, through partial matching rules. Both the original and generalized samples are quantified using

CHAPTER 2. LITERATURE REVIEW

the number of detectors in it. After quantitative comparisons in terms of the number of detectors needed for maximum coverage, it was proved that both detection schemes have the same detection capability within the context of several generation rules.

In comparison to our framework, theirs has several shortcomings. First, the binary string is not enough to distinguish normal and intrusive behaviors, while our framework provides a ‘suspicious’ class to represent the behaviors that may occur in the normal and intrusive runs simultaneously [LDZ05c]. Secondly, their framework has not well addressed the problem of sample incompleteness while we go further by evaluating the root cause of the problem. As indicated by the author, the generation rule may not be enough to overcome this problem. Lastly but not the least, unlike our framework, theirs is not useful for analyzing fundamental and practical problems faced by intrusion detection (see Section 1.2.1 in Chapter 1).

Base-rate Fallacy. In [Axe99], along with Bayes’ theorem, the base rate fallacy was applied to analyze the effectiveness of intrusion detection. In real deployments, normal behaviors are so much more frequent than intrusions that even their occasional misclassification can easily lead to an unacceptably high false alarm rate. Furthermore, the base-rate fallacy requires “*that in order to achieve substantial values of the Bayesian detection rate $P(\text{Intrusion}|\text{Alarm})$, we have to achieve a (perhaps in some cases unattainably) low false alarm rate [Axe99]*”. In other words, the higher false alarm rate will make a security site officer frustrated, so that soon the officer would start to ignore the (true and false) alarms altogether blindly.

The main contribution of the analysis is to emphasize the difficulty of intrusion detection caused by false alarms, but it has not shed light on the possible solutions to high false alarm rate. Our framework is complementary to this work by proposing a solution to reduce false alarms.

Analyzing Detection Rules. Manually crafted detection rules are always problematic because there is no quality guarantee on them. Along with the theorem prover ACL2, a framework was built to analyze and improve the quality of detection rules in [SKAF⁺04]. The framework consists of three components: an abstraction behavioral model, security properties, and intrusion detection rules. For every detection rule, a test is performed to determine whether it meets the specifications of behavior models and security properties. ACL2 was used to formalize an abstract behavioral

model and to carry out the verifications during the test. The experiments were performed on the detection rules in a specification-based intrusion detection system.

In our framework, we propose an automatic technique to build signatures from training audit trails for intrusion detection, in which the human errors are minimized. Nonetheless, their technique may be tuned to improve the quality of the training audit trails, which is the only vulnerable point in our framework.

Performance Evaluation. Feng et al. [FKF⁺03] analyzed and compared performance characteristics of several anomaly detection techniques (i.e., *N-gram*, *FSA*, *VtPath*, *callgraph*, *abstract stack* and *Var-gram*), all of which utilize the system-call related information in the call stack. The comparison was performed generally on their underlying principles in terms of false positives, detection ability, space requirements, convergence time, and runtime overhead. There is another work on performance evaluation. In [WFP99], an experimental comparison was also conducted in stide, t-stide, hidden Markov chain and RIPPER rule learning systems [Coh95]. Unfortunately, all the analysis and comparisons are not only specific for system-call based techniques, but also lacks a general framework for them. The lack of a general framework makes it difficult to analyze the generic problems faced by intrusion detection.

New Intrusion Detection Techniques. Several other intrusion detection frameworks (e.g., IDES [Den87], MADAM ID [LS00], STAT [VK99], EMERALD [PN97] etc.) have also been proposed in the literature. Their major motivations are to propose new techniques instead of analyzing and solving the generic problems. In contrast, we propose a general framework for intrusion detection, which can be utilized to argue about the principles of intrusion detection, to do the theoretical calculations of detection performance quantitatively, and then to analyze the generic problems systematically. Most importantly, the framework can also lead to a proposal for a novel intrusion detection technique naturally, in which SID and AID are treated uniformly.

In summary, most existing frameworks are either lacking capability for problem analysis or emphasizing the contributions of our framework in intrusion detection.

2.2 Model Generalization

We have evaluated the implications of model generalization for intrusion detection in Chapter 5. The purpose of model generalization (whose definition is given in Sec-

CHAPTER 2. LITERATURE REVIEW

tion 5.1) is to infer the anomalousness of a behavior, which is not in the current behavior models. However, it is two-edged for intrusion detection.

In AID techniques, more unknown normal behaviors will be recognized since the inference is based on the normal behavior models, and thus the detection performance is improved. But too little generalization will lead to false positives. Conversely, too much model generalization may overfeed the normal behavior models so that some intrusions are also included (Wagner and Soto [WS02]). The AID technique is thus cheated by these intrusions in normal behavior models, which are detected as ‘normal’.

In SID techniques, the generalization on existing intrusion signatures can alleviate the intrusion variation detection problem (Anchor et al. [AZGL02], Rubin et al. [RJM05]). However, if there is too little generalization on existing intrusion signatures, a clever attacker can use such generalization to generate intrusion variations to elude the SID system (Vigna et al. [VRB04], Rubin et al. [RJM04]). If there is too much generalization, the normal behaviors introduced by model generalization lead to false positives.

2.2.1 Extending Behavior Models

First, the intrusion signatures can be generalized to cover more intrusion variations. In [AZGL02], a session-based intrusion signature is represented as packet relationships in a so-called finite state transducer (FST). Then, the standard evolutionary programming approach was modified to use representations of finite state transducers, and then to generate the generalized intrusion signatures. In this approach, a fitness function is designed depending on false positives and false negatives. Then, the degree of generalization in an attack FST is optimized by reproduction and selection operators. At last, the generalized attack FST is utilized to detect the resource behaviors.

With the same objective, a methodology was presented in [RJM05] to construct more robust signatures automatically from session-based intrusion signatures. Two formal languages describe different properties of a given attack. First, a session signature (i.e., the first language) is used to describe the packet relations in an attack. The second language, an attack invariant language, tries to mine the properly generalized signature from a session signature. The optimization is performed manually to reduce false positives and false negatives [RJM05]. Essentially, model generalization

on intrusion signatures can solve the intrusion variations detection problem partially, but it is strictly limited by base-rate fallacy as just discussed.

Secondly, the normal behavior model of AID techniques can be generalized as well, and it can be done in several ways. Based on a distance function and a threshold (Denning [Den87], Eskin [EAP⁺02], Liao [LV02], Wang [WS04]), the instances in the existing audit trails are clustered, and the new instances are labeled by the existing instances in their clusters. In statistical methods for intrusion detection (MADAM ID [LS00], LERAD [MC02b], NIDES [JV93] etc.), the (statistical) resource usage profiles are mined from the existing audit trails. The novel instances are detected according to whether they fall into these profiles. In these two approaches, the existing audit trails are modeled *inexactly* to accommodate more resource behaviors within their profiles, and thus to achieve model generalization. In general, a certain extent of generalization is needed when building the behavior models, but these techniques didn't know the exact extent of model generalization applied. Under most existing evaluation scenarios, the extent of model generalization is implicitly chosen (or optimized) when the performance of an AID technique is optimized in detecting intrusions within specific audit trails. This philosophy of performance optimization on specific audit trails has left a blackhole for attacks to elude intrusion detection.

2.2.2 Eluding Intrusion Detection

There are several IDS evaluation methods proposed to utilize the false negatives introduced by model generalization. In AID techniques, mimicry attacks [WS02] are designed to cause false negatives in a host-based intrusion detection system. In order to make their attacking activities undetectable (i.e., false negatives), mimicry attacks try to mimic the normal behaviors in the normal behavior model. The success of mimicry attacks is caused by too much model generalization in the behavior models, and the inadequacy of attack knowledge leads to this overfitting model generalization. For example, suppose that at least five features are needed to distinguish between mimicry attacks and normal behaviors. Then, mimicry attacks could succeed if we use only four features for intrusion detection.

In SID techniques, false negatives outside an intrusion signature are also exploited by the mutated exploits [VRB04] and utilized by automatic attack signature generation [RJM04] to generate intrusion variations, and ultimately to avoid detection.

In [VRB04], the mutation operators in network layer, application layer and exploit layer were utilized to generate the mutants of an existing intrusion signature. The effectiveness of a mutant (note that it *may not* be an intrusion variation) was validated practically by the alert messages from RealSecure and Snort. In comparison, [RJM04] first manually built a set of sound inference rules, in which the soundness means that an intrusion variation will be generated if an inference rule is applied. Within the set of inference rules, there are pairs of expanding rules and shrinking rules. A shrinking rule reverses the effect of its corresponding expanding rule, and vice versa. The intrusion variations will be generated after applying all or part of these rules. From the perspective of model generalization, the mutation operators and inference rules first try to generalize existing intrusion signatures, and then to generate intrusion variations from the generalized signatures. Conversely, if an intrusion signature is generalized properly using methodologies in [RJM05, AZGL02], these elusive attacks will be covered by the signature and thus be detected.

In summary, too generous model generalization in AID will make mimicry attacks successful [WS02], while too stingy model generalization in SID will make some attack variations undetectable [RJM05, VRB04]. Unfortunately, there lacks a methodology to evaluate the utility of model generalization for intrusion detection. In our research, we make up for this deficiency by proposing an evaluation framework for model generalization. One objective of this evaluation is to identify the implications of model generalization on intrusion detection (or probably on other application fields as well).

2.3 Feature Selection

In intrusion detection, a set of features is used to record the characteristic values of a behavior in the training audit trails. However, two features may have the same distinguishability in intrusion detection. For example, *tcp_service_name* and *port_no* are equivalent among network packets. One feature may keep constant in the audit trails. For example, for all the outgoing traffic, *src_ip_addr* is always the IP address of the host. These features are useless in intrusion detection because they only waste the precious computation without any increase in behavior distinguishability. Feature selection is a method to identify and remove such features. We have applied a feature selection algorithm in our system to reduce the computation in Chapter 7.

CHAPTER 2. LITERATURE REVIEW

Feature selection is a critical step for intrusion detection [LS00]. However, most existing intrusion detection techniques utilize the features manually predefined by a security expert (Krügel et al. [KTK02], Forrest et al. [FPAC94], Mahoney et al. [MC02b]), and there are only a few reported work relating to feature evaluation for intrusion detection (Mukkamala and Sung [MS03], Lee and Stolfo [LS00]).

In [MS03], there are two methods for feature ranking. First, performance-based method is utilized to rank the features: one input feature F_i is deleted from the audit trails at a time, and the remaining audit trails is then used for the training and testing of a classifier. The performance of the resultant classifier is used to rank the feature F_i according to a predefined performance metric. Obviously, in order to evaluate every feature, the whole intrusion detection procedure, including training and detecting, should be run at a time. The second method is specific for support vector machines (Mitchell [Mit97]). Because the information about features and their contributions toward classification is hidden in the support vector decision function, the weight of every feature implicitly determines its rank among all features. In the second method, only the training audit trails are utilized to rank the features.

In the classification process of MADAM ID [LS00], the information gain, defined as the reduction of entropy that characteristics the ‘impurity’ of training audit trails, is utilized to rank the features. The features with larger information gain will be ranked higher. Furthermore, there is another process to construct temporal statistical features. The statistical patterns (i.e., frequent episodes) are first mined from normal and intrusive network audit trails respectively, and then these patterns thus generated are used as guidelines to construct temporal statistical features. Specifically, the intrusion-only statistical patterns are identified by comparing the patterns from normal audit trails with those from intrusive audit trails, and then they are parsed to construct statistical features accordingly with a heuristic ordering on the importance of ‘intrinsic’ features in a network record. Unfortunately, the statistical patterns may not reflect the audit trails completely, and there is no guarantee that all feature distinguishability will be used in the detection phase.

Feature selection is also performed in decision tree (Mitchell [Mit97]), which has been applied for intrusion detection as well (KDDCUP’99 [Elk99], Amor et al. [ABE04], Li and Ye [LY01]). In building decision trees, the feature selection is based on the

information gain by categorizing the training instances according to values of a feature. In every round, the feature with the largest information gain will be selected in building the decision tree. The stopping criteria is that the leaf node has a unique class label, or that all features are utilized before the current leaf node. There is one shortcoming in decision trees, which makes feature selection invalid. If two instances are identical except class labels, the decision tree will use all features to build a leaf node, and at last, the node is labeled using majority voting.

We propose a distance function to measure the relative distinguishing capability of a feature *only* using the training audit trails. If adding a feature does not increase the overall distinguishability, it is discarded while building the behavior models.

2.4 Behavior Signature Generation

Automatic signature generation is a challenging task for intrusion detection because it is difficult to guarantee the signature quality. In Chapter 8, we have proposed such an automatic system to generate signatures, the quality of which is guaranteed by our framework. As comparisons, we will talk about the related work on behavior signature generation in three aspects: *existing intrusion signatures*, *worm signatures* and *signatures for intrusion variations*.

2.4.1 Existing Intrusion Signatures

Intrusion signatures include the characteristic elements of intrusions/attacks. Existing intrusion detection systems provide different signature languages with various expressiveness, e.g., Snort [Roe99], RealSecure [Rea], Bro [Pax99], most of which are only used to express the network-based intrusions. For example, Snort signatures only represent network behaviors, and it is basically packet-based. There lacks a systematic and precise definition of intrusion signatures that can be widely applicable in both host-based and network-based scenarios. Moreover, the creation of these signatures is a tedious, manual process that requires detailed knowledge about software vulnerabilities, the application-specification settings and the intrusions themselves [SKAF⁺04]. Even worse, there does not exist a formal and general mechanism to guarantee the quality of the manually constructed signatures.

2.4.2 Worm Signatures

Because of the proliferation of worms, a lot of work (e.g., Honeycomb [KC04], Autograph [KK04], EarlyBird [SEVS04] and PolyGraph [NKS05]) has been proposed recently to generate worm signatures automatically.

A honeypot is a computer system sitting in the unused IP address space on the Internet, and its purpose is to attract and trap attackers who scan all IP address space and attempt to penetrate any existing computer system. Honeypots are able to capture the suspicious traffic of probe, attack or compromise activities, and hence it provides a major benefit in that one needs to look only at this traffic to generate signatures for these activities. Honeycomb [KC04] is an automatic tool to generate worm signatures, using pattern detection techniques and packet header conformance tests, from the suspicious traffic captured by honeypots. The inbound packets are aggregated with respect to TCP and/or UDP connections, which are kept as long as possible even after their terminations. If some stored connections have *the same destination port*, pattern detection will be utilized to identify signatures from the exchanged messages within these TCP and/or UDP connections. In [KC04], an existing pattern detection algorithm, Longest Common Substring, is applied to spot similarities in packet payloads. In their preliminary evaluation, the worm signatures for Slammer and CodeRed II have been produced precisely by their Honeycomb implementation.

Autograph [KK04] tries to automatically detect the signature(s) of any worm that propagates by randomly scanning IP addresses, without knowing the worm's payload or time of introduction in advance. Similar to Honeycomb, there are two stages to achieve the goal. First, a naive, portscan-based flow classifier is used as a heuristic to classify the inbound TCP traffic as suspicious or innocuous by assuming that all flows from port-scanning sources are suspicious. The suspicious TCP flows will be reassembled if necessary. Secondly, the payload in each suspicious flow is divided into smaller variable-length content blocks. The number of suspicious flows in which each content block occurs is counted to measure the prevalence of the content block. The content blocks are then ranked from most to least prevalent, and the most prevalent blocks become worm signatures to cover its pool of suspicious flows. In addition, Autograph incorporates a blacklisting strategy to suppress false positives.

The above two systems can be unified in the architecture level. First, a classifier is applied to classify the traffic into suspicious and innocuous flows. Secondly, the

CHAPTER 2. LITERATURE REVIEW

signature generation technique is devised to mine worm signatures from suspicious flows. PolyGraph [NKS05] utilizes the same deployment architecture with a different signature generation algorithm. In PolyGraph, a series of tokens is first extracted from the suspicious flows. Then, a clustering technique is applied to divide the suspicious flows into several clusters, and for every cluster, a worm signature, which may consist of several disjoint invariants, is generated to match every flow in it. The signature consists of several tokens in the style of conjunction signatures, token-subsequence signatures or Bayes signatures. One significant improvement in PolyGraph is that, instead of discarding innocuous flows as in Honeycomb and Autograph, the innocuous flows are utilized to minimize false positives.

With a specific requirement for high-speed implementation, Earlybird [SEVS04] is built for a packet level granularity rather than the flow level granularity in the above three systems. A content sifting algorithm is introduced in Earlybird to analyze network traffic for worm signatures. First, it breaks the payload of each packet into overlapping, fixed length blocks at every byte offset. The content block prevalence is estimated by a multi-stage filter, in which each block is hashed using the hash function of every stage, and the prevalence of the packet is based on all stages. Secondly, an address dispersion technique is applied to filter out the innocuous content blocks. A white-list strategy disallows the mined signatures that cause false positives.

In order to detect a worm, these four techniques try to identify *a single contiguous or disjoint invariant pattern* in the worm traffic, which will not lead to any false positive. Although there is no justification for the underlying assumption that there exist such unique pattern for every worm, experimental results have shown their effectiveness in building worm signatures. In addition, TaintCheck [NS05] can further help signature generation systems using information on the vulnerability and how it is exploited. In the first instance, it is able to produce a worm signature directly based on most significant overwrite value (which is a three-bytes string, at least experimentally). Secondly, it can also be utilized to classify the traffic into malicious and innocuous traffic as a traffic classifier, to verify the existing worm signatures, and to eliminate false alarms when the worm signatures are deployed.

Automatic classification on network traffic is easy to be poisoned by misleading attacks [PDL⁺06, NKS06], in which the normal network traffic is polluted to make the signature learning malicious. Furthermore, all the above techniques are proposed to

combat worms only, and their extensibility is very limited due to the dubious invariant patterns assumed within the packet payload. In comparison, our proposed signature generation system in Chapter 8 is hard to be poisoned since the feature ranges are collected from quality-guaranteed audit trails. However, from the past lessons on misleading attacks, we should pay much attention to strategies in signature analysis.

2.4.3 Signatures for Intrusion Variations

In order to perform intrusion detection evaluation, some signatures of unknown variations of an intrusion can be deduced from the known intrusion signatures. For example, some intrusion variations of an existing intrusion are generated automatically to elude the detection of Snort via mutate exploits [VRB04] and an automatic attack signature generator [RJM04]. The method will become invalid for totally new intrusions into a computing resource. Please refer to Section 2.2 for details.

Based on a formal definition of behavior signatures in our framework, we propose a realtime mechanism to generate them automatically, where the existing signature generation techniques, e.g. worm signature generation, could be incorporated into our mechanism. Importantly, the difficulty of intrusion variation detection is alleviated since the signatures of intrusion variations are captured in the detection phase.

2.5 Suppressing False Alarms

As described in Chapter 1, both SID and AID techniques are stuck with so many false alarms in the detection results that the effectiveness of an intrusion detection system is degraded. We have proposed a practical system to reduce false alarms. For comparison, we review existing efforts made to suppress false alarms in this section.

In [SB01], Sasha and Beetle proposed a concept of strict anomaly detection, where no false alarms will be triggered. The strict anomaly detection builds the model based on the formal specifications of network protocols. However, its effectiveness is doubtful since the specific protocol implementations in the Internet and computer systems are so diverse, and there are almost no implementations that follow protocol specifications exactly. In comparison, static analysis [WD01] is more realistic since it builds the behavior model from source/binary codes of specific implementations. Nonetheless, static analysis may be unable to detect mimicry attacks [WS02] (Chapter 3).

Other than proposing new techniques to suppress false alarms, there are three well-known approaches, which manipulate the alerts generated by an existing signature-based intrusion detection system. The *first* one is the multi-sensor fusion (Kruegel et al. [KMRV03]) in which the outputs of different IDS sensors are aggregated to produce a single alarm. This approach is based on the assumption that any single intrusion detection technique cannot classify a set of events as intrusion with sufficient confidence. The *second* approach uses alert/alarm correlations (Cuppens and Mieke [CM02], Ning [NCR02]), where the alerts/alarms with different timestamps and/or in different IDS sensors are correlated to build the attack scenarios. The *third* approach is to cluster the alarms to study their ‘root causes’ (Julisch [Jul03]), and then formulate specific rules to filter the corresponding alarms. However, these three solutions are mainly based on SID techniques in which the alerts provide more accurate information about intrusions. When they are applied to AID techniques, the accuracy of the alerts is unavoidably degraded, and under the worst scenario, it can become completely useless.

Up to date, there are only few techniques proposed to suppress false alarms generated by AID techniques. To the best of our knowledge, two AID techniques are proposed with simple false alarm suppressing schemes. In stide [FHSL96], a locality frame count scheme is proposed with a view to suppress false alarms by assuming that a true alarm will cause a cluster of anomalies in the audit trails. However, the assumption lacks a formal justification, and fails to account for dynamic behaviors of diverse computing resources. Similar to our proposed concept, PNrule [AJ00] uses a feedback mechanism (N-rules) to suppress false alarms in the behavior models (P-rules). In it, the suppressing scheme is also a supervised procedure. However, PNrule is limited by the knowledge and quality of the training audit trails, and there is no discussion about how to apply it into realtime scenarios for intrusion detection.

In this thesis, we propose a general mechanism to suppress the false alarms. Within the mechanism, a novel concept of anomaly context identification is introduced to make up for the knowledge deficiency in behavior models.

2.6 Summary

In our research, we try to conduct analysis, and then propose solutions for some generic problems of intrusion detection. In this chapter, we reviewed most existing

CHAPTER 2. LITERATURE REVIEW

efforts which have tried to address the same problems. However, we found that the existing efforts are not enough to solve all problems, and the lack of an efficient intrusion detection system partially indicates this fact. We will provide more detailed and pertinent comparisons between existing solutions and our proposed solutions in the following chapters.

Chapter 3

A Systematic Methodology for Intrusion Detection

As indicated in Chapter 1, our main objective is to analyze the generic problems in intrusion detection systematically, and hence to propose their solutions. In order to achieve it, a general intrusion detection framework, which incorporates both existing intrusion detection approaches, is necessary in our systematic methodology.

3.1 Background

Let us first imagine the following scenario. Bob saw a snake in the grassland yesterday, and asked the help of a snake expert to ascertain whether the snake is poisonous. Bob may describe the snake as follows, ‘it is a living animal’, ‘it is slim but very long’, ‘its length is between 1.5m and 2m’, ‘its diameter is around 10cm’, ‘its skin strip likes a zebra’, etc.. With more description, the expert can, little by little, identify the snake by the similarity to objects that he knew beforehand. Finally, the expert will come to a conclusion on whether the snake is poisonous with respect to his experiences.

In this identification process, The snake expert first extracts the characteristic features from Bob’s above description, namely, `category`, `isLiving`, `isSlim`, `isLong`, `Length`, `Diameter`, `SkinStrip`. He then compares them with these characteristic features of the known objects among his experiences. Before the identification, the snake expert has accumulated his ‘experiences’, especially about various snakes, as much as possible from various knowledge-sources (*e.g.*, *other experts*, *books*, *his own tests*). All these sources should be correctly labelled in advance. With more ‘experiences’, the identification becomes more accurate and efficient.

CHAPTER 3. A SYSTEMATIC METHODOLOGY FOR INTRUSION DETECTION

Similar to the identification process in human beings, the intrusion detection process is to identify the intrusions into computing resources. Mimicking the object identification process, we have proposed a formal framework for intrusion detection, in which we assume that *a set of features (i.e., object characteristics)* and *training audit trails (i.e., sources of expert experiences)* are known in advance. With respect to the predefined features, the behavior characteristics are extracted. The ‘experiences’ for intrusion detection are learned from the training audit trails, which are labeled beforehand. Table 3.1 lists the notations used in our framework.

Items	Meaning
FV	the feature vector, which consists of a set of features for intrusion detection.
F_i	the i^{th} feature in a feature vector FV .
I_{FV}^i	the i^{th} instance in audit trails, which instantiates all features in FV .
$Dom(F)$	the value domain of the feature F .
v_F, v_F^i	a value of the feature F , which is included in one instance in the training audit trails. $v_F \in Dom(F), v_F^i \in Dom(F)$.
R_F	a feature range $[v_F^1, v_F^2]$ of the feature F . $R_F \subseteq Dom(F)$.
$v(I_{FV}^i, F)$	the value of F in the instance I_{FV}^i .
$I(v_F, F)$	a set of instances that includes the feature value v_F of F in the training audit trails.
$I(R_F, F)$	a set of instances in which the values of F falls into R_F in the training audit trails.
$L(v_F)$	the NSA label of the value v_F .
$L(R_F)$	the NSA label of the feature range R_F .
$N(F)$	the normal feature subspace such that the NSA labels of all feature ranges within it are equal to ‘N’.
$S(F)$	the suspicious feature subspace such that the NSA labels of all feature ranges within it are equal to ‘S’.
$A(F)$	the attack feature subspace such that the NSA labels of all feature ranges within it are equal to ‘A’.
$\Omega(F)$	$\Omega(F) = N(F) \cup S(F) \cup A(F)$.
$F_{1...n}$	a n -order compound feature, such that its feature ranges are produced by the cartesian product of feature ranges of all its features $F_1, F_2, \dots, F_n$.
Sig^i	the i^{th} behavior signature for intrusion detection. It is the i^{th} feature range of $F_{1...n}$ if $FV = \{F_1, F_2, \dots, F_n\}$.
$I(Sig^i)$	a set of instances recognized by Sig^i .

Table 3.1: The notations in our framework.

3.2 A Comprehensive Framework

An intrusion detection system builds behavior models using a set of features of a resource, or a *feature vector* $FV = \{F_1, F_2, \dots, F_n\}$, where F_i is a feature in the feature vector. Every feature in the feature vector can be one of three types: a feature

CHAPTER 3. A SYSTEMATIC METHODOLOGY FOR INTRUSION DETECTION

associated with an instant of time (e.g., *the fields in the current packet*), or with a time interval (e.g., *the number of SYN packets within 2 seconds*), or with the context of the current event (e.g., *the system-call events in stide* [HFS98]). The context is defined over the timeline preceding the time point when the event in question happens.

The feature vector exists in most intrusion detection techniques explicitly and implicitly. There is a feature vector in each AID technique. For example, there are 6 context-sensitive features (i.e., system calls) in stide-6 [HFS98]. In ALAD [MC02a], the 5 features are: *source and destination IP address, destination port, TCP flags, keywords*. PAYL [WS04] utilizes the frequency information of 256 characters as features. In a SID technique, it is also very easy to identify the feature vector in any intrusion signature. For example, one SNORT signature [Roe99] and its feature vector are shown in example 3.1.

Example 3.1 *In the intrusion signature ‘alert tcp any any – > any 25 (msg:”SMTP expn root”; flags:A+; content:”expn root”; nocase; classtype:attempted-recon;)’, the features are: Protocol Type; Source Address; Source Port; Connection Type; Destination Address; Destination Port; TCP Flags; Characteristic Content; Content Case.*

To this end, we believe that a feature vector is common enough to represent most, if not all, inputs of intrusion detection systems.

In our framework, we also suppose that there are training audit trails, which are indexed by their timestamps, and labeled correctly beforehand. Corresponding to the feature vector, we collect a series of instances from the training audit trails, $\{I_{FV}^1, I_{FV}^2, \dots\}$. For every instance I_{FV}^i , there is also a **status** that indicates the label of audit trails where it is collected. For example, if an instance I_{FV}^i is collected from the audit trails produced by an intrusion ‘Nimda’, its **status** will be ‘Nimda’.

3.2.1 Basic Concepts and Notations

In this section, we will introduce several associated concepts of a feature F_k . In order to avoid cluttering expressions, we have dropped the subscript of F_k if not misleading.

Let us first define the type of the feature F . Based on the relations of its values, the feature F can be categorized into **nominal**, **discrete** or **continuous** one. A **discrete** or **continuous** feature is an ordinal feature type, such that its values can be ordered, and a distance can be determined between two values according to a distance metric.

CHAPTER 3. A SYSTEMATIC METHODOLOGY FOR INTRUSION DETECTION

A nominal feature is not quantifiable and hence do not possess any order amongst their values. In addition, a **discrete** feature has isolated points in a continuous spectrum of values. While the number of values for a **continuous** feature can be infinitely many, the number of **discrete** values is often few or finite. The feature vector FV could contain any number of **nominal**, **discrete**, and/or **continuous** features.

The manifestation of F in an instance I_{FV}^i is its **value**, denoted as v_F . Each of its values falls into its domain, which we call the **feature space** of F , denoted as $Dom(F)$. For example, for the feature ‘Destination Port’ in Example 3.1, one value $v_F = 25$ and $Dom(F) = [0, 65535]$. Besides the two common concepts, we introduce a new one, the **feature range** of F , denoted as R_F . It is an interval between two values of F , v_F^1 and v_F^2 , which includes all possible values between v_F^1 and v_F^2 in $Dom(F)$. The feature range concept can uniformize all features in FV , so that it is possible to analyze intrusion detection problems. The feature ranges could be manually defined or automatically collected from existing audit trails. In Section 6.1, existing feature values and their statuses have been utilized to construct feature ranges.

For a discrete or continuous feature, $R_F = [v_F^1, v_F^2]$. For a nominal feature, the concept of feature ranges does not apply as every value is independent. For uniformity, each value of a nominal feature is referred to as a feature range so that for a nominal feature F , $R_F = [v_F^i] = [v_F^i, v_F^i]$. If a value v_F^j is within the bounds of a feature range, we say that it falls within the feature range, denoted as $v_F^j \in R_F$. Moreover, the statuses of all instances, in which values of F fall within the feature range, are accumulated as the **status list** of the feature range.

Besides the above concepts of a feature F , there are also several notations related to the instances of the audit trails. For a feature $F \in FV$,

- $v(I_{FV}^i, F)$ is the value of the feature F in the instance I_{FV}^i .
- $I(v_F, F)$ is a set of instances for which the feature F has the value v_F ,

$$I(v_F, F) = \{I_{FV}^k | v_F = v(I_{FV}^k, F)\}$$

- $I(R_F, F)$ is a set of instances for which the values of F fall in R_F ,

$$I(R_F, F) = \{I_{FV}^k | v(I_{FV}^k, F) \in R_F\}$$

3.2.2 NSA Label

Definition 3.1 (NSA label of a value) *If a value v_F occurs only in the normal audit trails, it is assigned the label ‘normal’. If it occurs only in the attack audit trails, for example, an intrusion signature, it is labeled as ‘attack’. Otherwise, i.e., if it occurs in both normal and attack audit trails, it is labeled as ‘suspicious’. For brevity, the label ‘normal’, ‘suspicious’, or ‘attack’ is referred as the NSA label of v_F , denoted as $L(v_F) \in \{‘N’, ‘S’, ‘A’\}$, using the first letters of the labels.*

A value may exist in multiple instances of the training audit trails. Therefore, while a value in a specific instance is either normal or attack exclusively, its NSA label depends on all (*known*) instances in the training audit trails.

3.2.2.1 NSA labels of feature ranges

Based on the training audit trails, we can collect a set of feature ranges $\{R_F^1, R_F^2, \dots, R_F^m\}$, such that (1) $R_F^i \subseteq \text{Dom}(F)$, (2) there is no common feature value v_F , which falls in R_F^j and R_F^k at the same time ($j \neq k$), and (3) $I(R_F^i, F) \neq \emptyset$ ($i \geq 1$). Then, the concept of NSA label is extended to these feature ranges as follows. For R_F ,

$$\begin{aligned} L(R_F) = ‘N’ &\Leftrightarrow \forall i(v(I_{FV}^i, F) \in R_F \rightarrow L(v(I_{FV}^i, F)) = ‘N’) \\ L(R_F) = ‘A’ &\Leftrightarrow \forall i(v(I_{FV}^i, F) \in R_F \rightarrow L(v(I_{FV}^i, F)) = ‘A’) \\ L(R_F) = ‘S’ &\Leftrightarrow (\exists i(v(I_{FV}^i, F) \in R_F \wedge L(v(I_{FV}^i, F)) = ‘A’)) \\ &\quad \wedge \exists i(v(I_{FV}^i, F) \in R_F \wedge L(v(I_{FV}^i, F)) = ‘N’)) \\ &\quad \vee \exists i(v(I_{FV}^i, F) \in R_F \wedge L(v(I_{FV}^i, F)) = ‘S’)) \end{aligned}$$

After building the feature ranges for every feature in FV , the framework works on these feature ranges and audit trail instances for intrusion detection.

3.2.2.2 Feature Subspaces

Based on their NSA labels, the feature ranges of the feature F can be classified into three feature subspaces: $N(F)$, $S(F)$ and $A(F)$. Every feature subspace consists of the feature ranges, which have the same NSA label.

$$\begin{aligned} N(F) &= \{R_F^j \mid 1 \leq j \leq m, L(R_F^j) = ‘N’\} \\ S(F) &= \{R_F^j \mid 1 \leq j \leq m, L(R_F^j) = ‘S’\} \\ A(F) &= \{R_F^j \mid 1 \leq j \leq m, L(R_F^j) = ‘A’\} \end{aligned}$$

We also define $\Omega(F) = N(F) \cup S(F) \cup A(F) = \{R_F^1, R_F^2, \dots, R_F^m\}$ so that $\Omega(F)$ is a set of feature ranges found in the training audit trails. $\Omega(F)$ and $Dom(F)$ are strongly related while different. $Dom(F)$ is the domain of the feature F , where every feature range R_F^i in $\Omega(F)$ must fall in (i.e., $R_F^i \subseteq Dom(F)$). However, $\Omega(F)$ is not a complete partition of the whole value spectrum of $Dom(F)$. Based on different feature range construction methods, some values in $Dom(F)$ may not fall into any feature range in $\Omega(F)$ (refer to the example in section 3.2.4).

3.2.3 Compound Feature

Definition 3.2 (A compound feature) *A compound feature F_{12} is an ordered pair $\{F_1, F_2\}$, and $\Omega(F_{12})$ is a subset of the cartesian product of $\Omega(F_1)$ and $\Omega(F_2)$, such that each element in $\Omega(F_{12})$ represents at least one instance in the training audit trails. In other words, if an ordered pair $(R_{F_1}^a, R_{F_2}^b)$ is a compound feature range of F_{12} , then $I((R_{F_1}^a, R_{F_2}^b), F_{12}) \neq \emptyset$. For the sake of convenience, we will use the following notation to represent the compound operation $F_{12} = F_1 \times F_2$.*

The properties of a compound feature are similar to those of its component feature. (for discrimination, a single feature in FV is referred as an atomic one in the following description.) First, its feature ranges are exemplified by existing instances in the training audit trails. Based on the definition of *cartesian product*, any instance I_{FV}^i in $I((R_{F_1}^a, R_{F_2}^b), F_{12})$ will be recognized by feature ranges $R_{F_1}^a$ and $R_{F_2}^b$ as well. $I((R_{F_1}^a, R_{F_2}^b), F_{12}) = I(R_{F_1}^a, F_1) \wedge I(R_{F_2}^b, F_2)$. Secondly, its feature ranges are also mutually exclusive just like the feature ranges of its component features. This property is expressed in the following claim, which can be proved easily.

CLAIM: *The feature ranges of a compound feature are mutually exclusive. For two different feature ranges of a compound feature, $(R_{F_1}^a, R_{F_2}^b)$ and $(R_{F_1}^c, R_{F_2}^d)$, there is no such instance I_{FV}^i so that $I_{FV}^i \in I((R_{F_1}^a, R_{F_2}^b), F_{12})$ and $I_{FV}^i \in I((R_{F_1}^c, R_{F_2}^d), F_{12})$.*

Moreover, similar to atomic features, each feature range of F_{12} has its own NSA label as well, and $\Omega(F_{12}) = N(F_{12}) \cup S(F_{12}) \cup A(F_{12})$.

3.2.3.1 Compounding More Features

Since a compound feature shows the same behaviors as any of its component atomic features, we can treat the compound feature as an atomic one to build higher order compound features. Using this recursive procedure, the feature vector $FV = \{F_1, F_2, \dots, F_n\}$ can be converted into an equivalent n -order compound feature $F_{1\dots n}$.

3.2.4 An Illustrative Example

Let us assume that $FV = \{F_1, F_2, F_3\}$. The example instances of the feature vector FV are listed below in Table 3.2. The construction of feature ranges is listed as follows. The statuses of every value are first accumulated, and then their NSA labels are calculated. For discrete and continuous features F_2 and F_3 , the values are then sorted. The interval between two adjacent values is split equally, and every half is assigned the NSA label of its nearest value. For example, the two values 0.10 and 0.14 are adjacent, so $[0.10, 0.12]$ has the same label ‘S’ to the value ‘0.10’, and $(0.12, 0.14]$ has ‘A’. Finally, two adjacent feature ranges, which have the same NSA label, are merged. For the nominal feature F_1 , every value is a feature range by itself. More details on feature range construction are discussed in Section 6.1.

INDEX	F_1	F_2	F_3	STATUS
1	TCP	1	0.01	Normal
2	ICMP	2	0.04	Normal
3	NETBIOS	6	0.10	I_3
4	TCP	4	0.08	I_1
5	UDP	5	0.06	I_2
6	UDP	8	0.14	I_4
7	NETBIOS	6	0.10	Normal
8	UDP	7	0.02	Normal
9	UDP	8	0.14	I_2

Table 3.2: The example instances of the feature vector. We use ‘I’ to denote the status ‘intrusion’, and ‘Normal’ to represent normal behaviors.

We see that the value ‘TCP’ of the nominal feature F_1 can occur both in normal (instance 1) and intrusion¹ (instance 4) instances. Therefore, the NSA label of this value, which we call a range as this feature is non-numerical, is suspicious, and ‘ $Normal \& I_1$ ’ is its **status list**. We use the following shorthand to express this fact:

$$R_{F_1}^1 = TCP, Normal \& I_1 \Rightarrow 'S'$$

The feature ranges for every feature are listed as follows.

¹In this thesis, the terms ‘intrusion’ and ‘attack’ will be used alternatively to represent malicious activities to a protected computing resource.

CHAPTER 3. A SYSTEMATIC METHODOLOGY FOR INTRUSION DETECTION

- For F_1 ,
 - $R_{F_1}^1 = TCP, Normal \Rightarrow 'S'$
 - $R_{F_1}^2 = ICMP, Normal \Rightarrow 'N'$
 - $R_{F_1}^3 = UDP, Normal \& I_2 \& I_4 \Rightarrow 'S'$
 - $R_{F_1}^4 = NETBIOS, Normal \& I_3 \Rightarrow 'S'$
- For F_2 ,
 - $R_{F_2}^1 = [1, 2], Normal \Rightarrow 'N';$
 - $R_{F_2}^2 = [3, 5], I_1 \& I_2 \Rightarrow 'A';$
 - $R_{F_2}^3 = [6, 6], Normal \& I_3 \Rightarrow 'S';$
 - $R_{F_2}^4 = [7, 7], Normal \Rightarrow 'N';$
 - $R_{F_2}^5 = [8, 8], I_2 \& I_4 \Rightarrow 'A';$
- For F_3 ,
 - $R_{F_3}^1 = [0.01, 0.05], Normal \Rightarrow 'N';$
 - $R_{F_3}^2 = (0.05, 0.09], I_1 \& I_2 \Rightarrow 'A';$
 - $R_{F_3}^3 = (0.09, 0.12], Normal \& I_3 \Rightarrow 'S';$
 - $R_{F_3}^4 = (0.12, 0.14], I_2 \Rightarrow 'A';$
- $F_{23} = F_2 \times F_3$
 - $R_{F_{23}}^1 = R_{F_2}^1 \times R_{F_3}^1, Normal \Rightarrow 'N';$
 - $R_{F_{23}}^2 = R_{F_2}^2 \times R_{F_3}^2, I_1 \& I_2 \Rightarrow 'A';$
 - $R_{F_{23}}^3 = R_{F_2}^3 \times R_{F_3}^3, Normal \& I_3 \Rightarrow 'S';$
 - $R_{F_{23}}^4 = R_{F_2}^4 \times R_{F_3}^4, Normal \Rightarrow 'N';$
 - $R_{F_{23}}^5 = R_{F_2}^5 \times R_{F_3}^4, I_2 \& I_4 \Rightarrow 'A';$
- $F_{123} = F_1 \times F_{23}$
 - $R_{F_{123}}^1 = R_{F_1}^1 \times R_{F_{23}}^1, Normal \Rightarrow 'N';$
 - $R_{F_{123}}^2 = R_{F_1}^2 \times R_{F_{23}}^1, Normal \Rightarrow 'N';$
 - $R_{F_{123}}^3 = R_{F_1}^1 \times R_{F_{23}}^2, I_1 \Rightarrow 'A';$
 - $R_{F_{123}}^4 = R_{F_1}^3 \times R_{F_{23}}^2, I_2 \Rightarrow 'A';$
 - $R_{F_{123}}^5 = R_{F_1}^4 \times R_{F_{23}}^3, Normal \& I_3 \Rightarrow 'S';$
 - $R_{F_{123}}^6 = R_{F_1}^3 \times R_{F_{23}}^4, Normal \Rightarrow 'N';$
 - $R_{F_{123}}^7 = R_{F_1}^3 \times R_{F_{23}}^5, I_2 \& I_4 \Rightarrow 'A';$

The difference between $Dom(F)$ and $\Omega(F)$ is obvious in this example. $Dom(F_1) = \{TCP, UDP, ICMP, NETBIOS, DHCP, \dots\}$, which includes most popular protocols, but $\Omega(F_1)$ includes only four protocols in the example instances: TCP, ICMP, UDP and NETBIOS. It is also shown in the feature F_3 : $Dom(F) = [0, 1.0]$ but the value 0.37 does not fall into any feature range of F_3 .

The feature subspaces of $F_{1\dots n}$ are: $N(F_{123}) = \{R_{F_{123}}^1, R_{F_{123}}^2, R_{F_{123}}^6\}$, $S(F_{123}) = \{R_{F_{123}}^5\}$, and $A(F_{123}) = \{R_{F_{123}}^3, R_{F_{123}}^4, R_{F_{123}}^7\}$.

3.2.5 Behavior Signature

Definition 3.3 (behavior signature) Suppose that there is a feature vector for intrusion detection $FV = \{F_1, F_2, \dots, F_n\}$, and the feature ranges of every feature are determined beforehand. A **behavior signature** Sig^i includes a feature range of $F_{1\dots n}$, its NSA label and its status list, denoted as $[R_{F_{1\dots n}}^i, L(R_{F_{1\dots n}}^i), StatusList]$.

Example 3.2 Take the same scenario as in Table 3.2, one behavior signature from $R_{F_{123}}^2 (= R_{F_1}^2 \times R_{F_2}^1 \times R_{F_3}^1)$ is $[(ICMP, [1, 2], [0.01, 0.05]), 'N', 'Normal']$.

As indicated in the above definition, every behavior signature² represents a state of the resource. As a special case with $n = 1$, the signature can be a value in the training audit trails. For example, the TCP port '139' is at some point determined as 'attack' due to the widespread worm 'Nimda'.

²For brevity, 'behavior signature' will be simplified as 'signature' in the following description.

All signatures in $\Omega(F_{1\dots n})$ constitute the behavior models. In AID, only the normal behavior model $N(F_{1\dots n})$ is utilized, but SID identifies intrusions based on the intrusive behavior model $A(F_{1\dots n})$. In our method USAID (Chapter 6), we use the whole behavior models $\Omega(F_{1\dots n})$ for intrusion detection.

Mathematically, the maximal number of signatures is $\prod_{i=1}^n |\Omega(F_i)|$ using the cartesian product. In the above example of section 3.2.4, the maximal number of signatures is $\prod_{i=1}^3 |\Omega(F_i)| = 4 * 5 * 3 = 60$, but, in practice, there are only 7 signatures for F_{123} . This is because the number of signatures is also limited by the number of existing instances (i.e., known knowledge). This leaves some *new* instances unrecognizable. Model generalization, addressed in the following section, is a solution to recognize more (*new*) instances using limited knowledge.

3.2.5.1 Model Generalization among Signatures

In general, all behaviors in the training audit trails can be represented *precisely* in our framework, but the training audit trails are always limited by known knowledge, and hence the set of behavior signatures. To this end, model generalization is always applied for intrusion detection [Den87, LS00, AZGL02, RJM05] such that the signatures in the behavior models are able to recognize not only the instances in the training audit trails, but also the instances outside the training audit trails. In our framework, there are three phases in which we can apply three levels of generalization: 1) **constructing feature ranges**, 2) **building behavior models** and 3) **detecting an instance using its similarity to a signature**. The generalization in different levels has different assumptions (see also Section 5.1).

- L1: [Generalizing feature ranges]. In a user-defined *collection strategy*, the feature ranges of every feature are generalized to cover more unknown feature space. In the example of section 3.2.4, the feature range $R_{F_3}^3 = (0.12, 0.14]$ can be generalized to $R_{F_3}^3 = (0.12, 0.50]$, and hence to cover any value between 0.14 and 0.50 while it is not recognized by the signatures in the behavior models.
- L2: [Generalizing the behavior models]. Using the *grouping or mutation* operations (or other operations), especially on two signatures with the same NSA label, some signatures that are not in the training audit trails are augmented into the behavior models. To some extent, the effect of L2 generalization is to ignore several features in building such signatures.

- L3: [Generalizing signatures during detection]. With a *distance function*, the generalization introduces some signatures that are not in the training audit trails but within a distance threshold to the existing signatures.

In a converse manner, to meet the necessity of model generalization under different scenarios, we can adjust and configure these three levels of generalization according to their usefulness for intrusion detection.

3.2.5.2 Detecting Instances via Signatures

Suppose that the behavior models of a computing resource consist of a series of signatures, $Sig^1, Sig^2, \dots, Sig^m$. For any signature in the behavior models, $Sig^i = (R_{F_1}^{i_1}, R_{F_2}^{i_2}, \dots, R_{F_n}^{i_n})$, and we further denote the set of instances recognized by Sig^i as $I(Sig^i)$. According to the definition of the compound feature $F_{1\dots n}$,

$$I(Sig^i) = I(R_{F_1}^{i_1}, F_1) \cap I(R_{F_2}^{i_2}, F_2) \cap \dots \cap I(R_{F_n}^{i_n}, F_n)$$

Thus, to detect an instance, for example I_{FV}^j ,

$$I_{FV}^j \in I(Sig^i) \iff \bigwedge_{k=1}^n v(I_{FV}^j, F_k) \in R_{F_k}^{i_k}$$

In words, in order to detect I_{FV}^j , all of its feature values must fall into their corresponding feature ranges in a signature. Then, the relation between the NSA label of a signature and the status of the instance is utilized for intrusion detection.

3.3 Intrusion Detection Problem

The general task of intrusion detection is to build the behavior model of a computing resource, and then to inspect the audit trails of the computing resource to detect possible intrusions using the behavior models. In our framework, the intrusion detection problem is formally defined as follows.

Definition 3.4 (Intrusion Detection Problem) *Suppose that there exist training audit trails, test (or realtime) audit trails, and a feature vector $FV = \{F_1, F_2, \dots, F_n\}$. The instances in the training audit trails are labeled correctly in advance. There are two phases in intrusion detection:*

CHAPTER 3. A SYSTEMATIC METHODOLOGY FOR INTRUSION DETECTION

- *In the training phase, the behavior models are built from the training audit trails.*
 - (1) **Extract** instances from the training audit trails, $I_{FV}^1, I_{FV}^2, I_{FV}^3, \dots$, and label them using statuses of instances in the training audit trails;
 - (2) **Learn** the feature ranges of all features in FV , $\Omega(F_1), \Omega(F_2), \dots, \Omega(F_n)$, and the behavior models from all instances, which are constituted with a series of signatures, $Sig_{FV}^1, Sig_{FV}^2, \dots, Sig_{FV}^m$.
- *In the detecting phase, the intrusions in the test audit trails are detected and/or identified.*
 - (1) **Extract** an instance I'_{FV} from these audit trails one at a time;
 - (2) With respect to each signature Sig_{FV}^i in $\Omega(F_{1\dots n})$, if $\bigwedge_{k=1}^n (v(I'_{FV}, F_k) \in R_{F_k}^{i_k})$, the instance I'_{FV} will be **identified** by the status list of Sig_{FV}^i . Otherwise, it will be **detected** as ‘normal’ or ‘anomaly’ w.r.t. different detection approaches (e.g., AID, SID) because there lacks knowledge about the ‘new’ instance in the behavior models.

There is no ambiguity among the detection results in the above definition because only one signature in the behavior model identifies an instance. Due to the same reason, the detection procedure stops if a new instance is identified. It saves the time to check whether remaining signatures are able to identify it.

3.4 Venn Diagram Representation

Venn Diagram [RW05] can be applied here to clearly state the relations of the NSA feature subspaces in our framework. We use 2-Venn Diagrams. Suppose that there are three instance sets from the audit trails of a protected computing resource:

- I_u , all the audit trail instances that the protected resource could generate.
- I_n , all the normal instances in the training audit trails, and
- I_a , all the attack instances in the training audit trails.

Their relations are illustrated in Figure 3.1. Based on the three instance sets, the whole area in the above Venn diagram is split into four regions.

Figure 3.1: The Venn diagram of the audit trail instance sets. $I_n \subseteq I_u$ and $I_a \subseteq I_u$.

(1) $I_X = I_n \cap I_a$

A region where the known instances can occur in both normal and attack runs of the computing resource. Every instance in it have a *suspicious* signature.

(2) $I_Y = I_n - I_X$

A region where the known instances are generated by normal runs of the resource. Every instance in this region has a *normal* signature.

(3) $I_Z = I_a - I_X$

A region where the known instances are generated by attack runs of the resource. Every instance in it has an *attack* signature.

(4) $I_W = I_u - (I_n \cup I_a)$

A region including all unknown (*normal and attack*) instances, which may be generated in future runs of the resource.

Mapping them to our framework, the signatures in the feature subspace $S(F_{1...n})$ are able to identify the instances in I_X , $N(F_{1...n})$ identifies I_Y and $A(F_{1...n})$ identifies I_Z . We cannot identify any instance in I_W but it is detected as an ‘**anomaly**’, which is caused by an unknown intrusion or an unknown normal behavior. In addition, the purpose of model generalization is to identify instances in I_W using the signatures in $\Omega(F_{1...n})$. Under the ideal situation, $I_W = \emptyset$.

3.5 Desirable Properties

We believe that the framework thus introduced has a number of very desirable properties that make it attractive for intrusion detection, such as,

- **A formal definition of behavior signatures:** the normal and intrusive behavior signatures are defined naturally without any ambiguity.

- **Uniqueness:** signatures in the behavior models are mutually exclusive, and thus there is no ambiguity in detecting an instance.
- **Quantified behavior models/spaces:** the behavior spaces/models can be quantified with the numbers of signatures within them. We can represent the detection performance and analyze generic problems in a quantified manner.
- **Adjustable quality factors:** This framework can describe the known knowledge pretty precisely and extend the knowledge via a configurable extent of model generalization. The quality of the ultimate behavior models depends on the extent of model generalization. Therefore, we can configure the generalization level to achieve a certain quality in the behavior models.
- **Distribution-free:** our framework is rooted in the concept of feature range, which does not depend on statistical and frequency information. It is important because it is difficult to get the explicit distribution of network/host audit trails. Nonetheless, it does not exclude any possibility of using the distribution information in our framework. In our implementations (Chapters 5-8), the frequency information is already used in building feature ranges.
- **Incremental:** it is easy to append a new feature in the feature vector, and it is also easy to append a new feature range to an existing feature dynamically. The computing world is evolving continuously nowadays, it is important to adjust the features utilized for intrusion detection due to the change of audit trails, attack strategies, security tools (e.g., AnitSpyware, AntiVirus softwares) etc..
- **Distributed:** as mentioned earlier, a feature can also be an output of a meta-intrusion detection system. Thus, different features can be deployed distributedly and/or hierarchically to detect intrusions collaboratively.
- **Privacy protection:** The values in an instance are represented by feature ranges. In this way, the privacy information in the values has been protected. Moreover, the feature ranges can be indexed to provide more privacy protection.

In this thesis, we will apply these properties to achieve our main objective. That is, they are used to analyze generic problems, and then to propose solutions for them.

3.6 Using the Framework

In the following chapters, we will use the framework to design, analyze and evaluate intrusion detection systems.

3.6.1 Its Applications

So far, we have proposed and evaluated the following applications of our framework.

- *Analyzing generic problems theoretically (Chapter 4)*

We will use the properties of ‘*uniqueness* and *quantified behavior models/spaces*’ to analyze the fundamental and practical problems theoretically.

- *Evaluating model generalization experimentally (Chapter 5)*

We will use the property of ‘*adjustable quality factors*’ to evaluate the utility of model generalization for intrusion detection.

- *Proposing novel intrusion detection techniques (Chapter 6 and 7)*

We will use the properties of ‘*uniqueness*, *distribution-free*, and *distributed*’ to design a new technique, and it results in very good detection performance. Furthermore, using the ‘*incremental*’ property, the computational burden of the detection process could be reduced.

- *Proposing an automatic signature assembly line (Chapter 8)*

We will use the properties of ‘*distribution-free*, *incremental*, *privacy protection*, *uniqueness*, *formal definition of signatures*’ to build signatures automatically.

In order to evaluate them experimentally, we have predefined a feature vector, and have selected the following dataset as the training audit trails in our experiments.

3.6.2 The Experimental Dataset

The dataset are from KDD CUP 1999 contest, which is typical for network intrusion detection.³ This dataset meets the requirements of our framework: *labeled audit trails and an intrusion-specific feature vector*. Its specification is listed as follows. There are 41 features, 4898431 instances in the training audit trails, 311029 instances in the test audit trails. The value precision of discrete features is $\varepsilon_d = 1$, and the value precision of continuous features is $\varepsilon_c = 0.01$. For a detailed description of the dataset, please refer to [Elk99].

Table 3.3 lists all the features used in our experiments, and Table 3.4 enumerates the known and unknown intrusions in the audit trails. The known intrusions in the

³Most intrusion detection techniques need a supervised dataset. For example, in a SID technique, it is a dataset with only intrusive behaviors, and in an AID technique, it is a normal dataset with only normal behaviors. However, it is difficult to collect such a dataset. To our knowledge, only this supervised KDD CUP dataset, which is publicly available, meets the requirements of our framework.

CHAPTER 3. A SYSTEMATIC METHODOLOGY FOR INTRUSION DETECTION

training/test datasets are used to evaluate the identification capability of an intrusion detection technique, and the unknown intrusions in the test dataset are used to pinpoint its capability for anomaly detection.

One may argue that the dataset has problematic issues, such as poor distribution, inconsistent hypotheses [SS04] such that it is not proper to evaluate intrusion detection systems. However, the issues will not affect the analysis and evaluation using our framework. This is because our framework does not depend on the distribution and statistical information of the dataset.

TYPES (41)	FEATURES
nominal (9)	protocol type, service, flag, land, logged_in, root shell, su_attempted, is_hot_login, is_guest_login
discrete (15)	duration, src_bytes, dst_bytes, wrong_fragments, urgent, hot, num_failed_logins, num_compromised, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, count, srv_count
continuous (17)	error_rate, srv_error_rate, error_rate, srv_error_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_error_rate, dst_host_srv_error_rate, dst_host_error_rate, dst_host_srv_error_rate

Table 3.3: The feature vector from connection records.

CATEGORY	MEANING	KNOWN INTRUSIONS	NEW INTRUSIONS
normal	normal	—	—
probe	probe	ipsweep, nmap, portsweep, satan	mscan, saint
DOS	denial of service	back, land, neptune, pod, smurf, teardrop, warezmaster	apache, mailbomb, processtable, udpstorm
U2R	user to root	buffer_overflow, loadmodule, multihop, perl, rootkit	httptunnel, sqlattack, xterm, ps
R2L	remote to local	ftp_write, guess_passwd, httptunnel, imap, multihop, phf	named, sendmail, snmpgetattack, snmpguess, worm, xlock, xsnoop

Table 3.4: Intrusions in the training and test datasets.

In our experiments, we have discarded some illegal instances in the training and test audit trails. In the test audit trails, we found that an error occurs when it detected the instances (instances 136489 and 136497) with ‘TCP’ protocol type and ‘ICMP’ service. Such combination is illegal in the Internet protocol specifications, so they have been discarded in our experiments. Moreover, in the training audit trails, there is an intrusion ‘spy’ that is not documented properly (in instances 1381226, 1381227). Its signature has been generated in our experiments.

3.7 Summary

In this chapter, we have laid out the basic framework for our proposed intrusion detection system. The framework attempts to capture the object identification process used by human beings and prepares the foundation for a systematic methodology for intrusion detection, that will be exploited in the subsequent chapters. The various concepts related to intrusion detection as well as the tasks that we set out to complete in this thesis are formalized by mapping them out into this framework. We have enumerated the useful properties of the proposed framework and listed a number of useful applications of the framework. Finally, we have described in detail the dataset (from KDD cup 1999) that is used in our experiments.

Chapter 4

Problem Analysis

As discussed in Chapter 1, intrusion detection has been stuck with its fundamental and practical problems nowadays. It is critical to solve these problems before intrusion detection can be applied effectively. We share the views of John McHugh that “further significant progress for intrusion detection will depend on the development of an underlying theoretical basis for it [McH01]”, but at present, “there is no underlying theory that relates detection approaches to detections or that allows useful predictions to be made concerning the relative powers of various techniques [McH01]”. In our research, we try to develop such underlying theoretical basis for intrusion detection. Briefly, we quantify the detection performance to analyze the generic problems for intrusion detection.

4.1 Preliminaries

There are two behavior spaces in a computing resource for intrusion detection (Figure 4.1): *normal behavior space* and *intrusive behavior space*, and they are complementary to each other. Any behavior in the normal behavior space comes from a normal run of the computing resource, and any behavior in the intrusive behavior space comes from an intrusive run. Conceptually, SID is based on only the knowledge of the intrusive behavior space, and AID is based on only the knowledge of the normal behavior space (Chari et al.[CC03]). Perfect detection of intrusions can be achieved if and only if we have a complete behavior model of any one of the two behavior spaces, because, ideally, what is not bad is good and vice versa.

However, it is difficult to model such behavior spaces completely and correctly in reality due to several reasons: (1) the known knowledge is always limited, (2) the modeling techniques may lead to incorrect inferences, and (3) the phenomenon of

4.1.a: The intrusive behavior model in SID.

4.1.b: The normal behavior model in AID.

Figure 4.1: Behavior spaces and models for intrusion detection.

concept drifting always exists in many scenarios. Figure 4.1 illustrates the behavior models for SID (i.e., *intrusive behavior model*) and for AID (i.e., *normal behavior model*) in the real applications (Chari and Cheng[CC03]). As the figures indicate, there exist model errors in the behavior models for SID techniques as well as AID ones. These model errors are called the *inaccuracy* in the behavior models. For example, a part of intrusive behavior model falls into normal behavior space. Furthermore, the intrusive behavior model cannot cover all intrusive behavior space, and the normal behavior model cannot cover all normal behavior space either. This is referred to as the *incompleteness* of the behavior models.

In summary, there are two quality factors in every behavior models, namely *inaccuracy* and *incompleteness*. In SID techniques (Figure 4.1.a), model inaccuracy in the intrusive behavior model will lead to *false positives* (FP) and model incompleteness in it will lead to *false negatives* (FN). In contrast, due to the different detection mechanisms, model inaccuracy in the normal behavior model will lead to FNs and model incompleteness in it will cause FPs in AID techniques (Figure 4.1.b). In order to develop a useful intrusion detection system, it is critical to precisely quantify the impact of these two factors on intrusion detection.

In the following sections, we will argue about the relations between the quality factors and intrusion detection performance.

4.2 Quantifying Behavior Spaces/Models

We will focus on a compound feature F in our analysis since all the features in the feature vector can be integrated into a higher-order compound feature (Chapter 3). We

CHAPTER 4. PROBLEM ANALYSIS

also assume that every behavior signature has the same significance in our analysis, i.e., its frequency information is not utilized in this analysis. This is because our analysis objective is to identify the general relations between the quality factors and the detection performance rather than to quantify the exact influence of these quality factors on the detection performance.

Feature Subspaces. First, let us define the feature subspaces under two different scenarios. In the *ideal* scenario, which is a hypothetical scenario that assumes *complete* behavior knowledge of a computing resource, the three feature subspaces, $N_i(F)$, $A_i(F)$ and $S_i(F)$, are determined completely without any misclassification. In the *real* scenario where we possess only *partial* behavior knowledge, the three feature subspaces are defined as $N_r(F)$, $A_r(F)$, and $S_r(F)$.

4.2.1 Model Inaccuracy

To quantify model inaccuracy, we use several subsets of signatures, which are misclassified due to the inaccuracy: $NA_1(F)$, $NS_1(F)$, $SN_1(F)$, $SA_1(F)$, $AN_1(F)$, and $AS_1(F)$. Every subset name is generally defined as follows: the first letter denotes the actually assigned feature subspace, and the second letter is the ideal feature subspace. The subscript ‘1’ indicates that the subset is caused by the first model quality factor (i.e., *model inaccuracy*). For example, the signatures in $NA_1(F)$ are assigned to be in $N_r(F)$ due to the inaccuracy defect, but ideally they should be in $A_i(F)$.

4.2.2 Model Incompleteness

Model incompleteness can also be quantified using several signature subsets. First, because of model incompleteness, some unknown signatures have not occurred in existing audit trails but are likely to occur in future audit trails. These *future* normal, suspicious, and attack signatures are denoted as $N_f(F)$, $S_f(F)$ and $A_f(F)$ respectively. Secondly, by increasing completeness, i.e., with more known information about resource behaviors, some signatures in $N_r(F)$ and $A_r(F)$ will move to $S_i(F)$. The change is quantified by the following two signature subsets, $NS_2(F)$ and $AS_2(F)$. $NS_2(F)$ represents the subset of signatures in $N_r(F)$ which change to $S_i(F)$, and $AS_2(F)$ is the subset of signatures in $A_r(F)$ which change to $S_i(F)$. Also, the subscript ‘2’ indicates the second model quality factor (i.e., *model incompleteness*).

CHAPTER 4. PROBLEM ANALYSIS

Conversely, these signature subsets embody both quality factors in the behavior models. To make it more clear, the following equations are built to represent the relations among the signature subsets listed above.

$$\begin{aligned} N_i(F) &= N_r(F) - NA_1(F) - NS_1(F) - NS_2(F) + SN_1(F) + AN_1(F) + N_f(F) \\ A_i(F) &= A_r(F) - AN_1(F) - AS_1(F) - AS_2(F) + SA_1(F) + NA_1(F) + A_f(F) \\ S_i(F) &= S_r(F) - SN_1(F) - SA_1(F) + NS_1(F) + NS_2(F) + AS_1(F) + AS_2(F) + S_f(F) \end{aligned}$$

where, ‘+’ and ‘-’ denote set union and difference operations.

4.3 Problem Analysis

In this section, we first quantify the detection performance based on principles laid out so far, and then analyze the generic problems affecting intrusion detection. As mentioned before, our focus is to quantify the detection performance of an intrusion detection system using the feature F and its behavior signatures.

	INTRUSION	NORMAL
ALARM	$TP(F)$: True Positive	$FP(F)$: False Positive
NON-ALARM	$FN(F)$: False Negative	$TN(F)$: True Negative

Table 4.1: Detection results v.s. true classes. The rows correspond to the detection outcomes, and the columns correspond to the true classes of resource behaviors in question.

Two Performance Metrics. In our analysis, the two common used performance metrics are applied to represent the detection performance: the detection rate $DR(F)$ and the false alarm rate $FAR(F)$. Using the four categories within the detection results as shown in Table 4.1, both metrics can be represented :

$$DR(F) = \frac{|TP(F)|}{|TP(F)| + |FN(F)|} \quad FAR(F) = \frac{|FP(F)|}{|TP(F)| + |FP(F)|}$$

where, $|\dots|$ represents of the size of a signature set.

In our following analysis, there are three parameters deserving our mention here. First, two parameters x and y are used to compare the sizes of the behavior models within the *ideal* scenario while it is impossible to get them in the real scenario. Specifically, $|S_i(F)| = x * |N_i(F)|$ and $|A_i(F)| = y * |N_i(F)|$. Secondly, we assume that a fraction α of the signatures labeled as ‘suspicious’ in the behavior model will be detected as ‘attacks’, and hence a fraction $1 - \alpha$ of them will be detected as ‘normal’.

4.3.1 Signature-based Intrusion Detection

Definition 4.5 (signature-based intrusion detection) *It is an intrusion detection problem, that uses only an intrusion behavior model $A_r(F)$ (a.k.a. the intrusion signature base) derived from intrusion audit trails only. In the detection phase, the behaviors, which do not match any signature in $A_r(F)$, are regarded as ‘normal’ ones.*

Figure 4.2: The intrusive behavior model $A_r(F)$ in SID techniques, where (1) $e + f + g = 1$, (2) $S_i(F) = x * |N_i(F)|$, $A_i(F) = y * |N_i(F)|$, (3) $|AS_1(F)| + |AS_2(F)| = f * |A_r(F)|$, (4) $|AN_1(F)| = g * |A_r(F)|$.

In SID, $S_r(F) = \emptyset$, and $N_r(F) = \Omega(F) - A_r(F)$. To measure model inaccuracy and model incompleteness in $A_r(F)$, we introduce three additional parameters e , f , and g representing fractions. $e * |A_r(F)|$ is the number of signatures that are correctly categorized into $|A_r(F)|$, $f * |A_r(F)|$ is the number of signatures that are categorized incorrectly due to model incompleteness, and $g * |A_r(F)|$ is the number of signatures that are categorized incorrectly due to model inaccuracy.

From Figure 4.2, the four detection categories in Table 4.1 are deduced¹:

$$\begin{aligned}
 |TP(F)| &= |A_r(F)| - (|AS_1(F)| + |AS_2(F)|) * (1 - \alpha) - |AN_1(F)| \\
 &= |A_r(F)| * (e + f\alpha) \\
 |FP(F)| &= (|AS_1(F)| + |AS_2(F)|) * (1 - \alpha) + |AN_1(F)| \\
 &= |A_r(F)| * (f(1 - \alpha) + g) \\
 |FN(F)| &= |A_f(F)| + |S_f(F)| * \alpha \\
 |TN(F)| &= |N_i(F)| + |S_f(F)| * (1 - \alpha) - |AN_1(F)|
 \end{aligned}$$

¹As a retrospection, we assume that a fraction α of the feature ranges labeled as ‘suspicious’ in the behavior models will be detected as ‘attacks’.

Then, we can quantify the detection performance as follows:

$$DR(F) = \frac{|A_r(F)|}{|N_i(F)|} \times \frac{e + f\alpha}{y + x\alpha} \quad FAR(F) = g + f(1 - \alpha)$$

In general, the suspicious feature subspace $S_i(F)$ can be empty if a proper feature vector is selected. Conversely, if all possible features are selected but $S_i(F) \neq \emptyset$, the soundness of SID cannot be guaranteed [McH01]. In addition, if $N_i(F)$ is stable, the detection rate will be degraded with more vulnerabilities (i.e., the increase of y).

Lastly, based on the quantified detection performance (i.e., $DR(F)$ and $FAR(F)$), we can analyze the existing hard problems in SID techniques.

$\mathcal{P}_S^1$: Incapable of detecting new intrusions

Even under a perfect scenario, i.e., $f = g = 0$, the detection rate is determined by the current and potential signature bases $A_r(F)$ and $A_i(F)$. As $A_i(F)$ is relatively stable, the signatures in $A_r(F)$ dominate the detection rate. Thus, the incapability of detecting new intrusions is caused by the limited known intrusions from existing audit trails, and the situation becomes worse with respect to the quality problem of the intrusion behavior model ($e + f\alpha \leq 1$).

$\mathcal{P}_S^2$: Signature updating problem

To improve the detection performance, we should first improve the detection rate, and thus include more intrusion signatures in $A_r(F)$ (cf., $\mathcal{P}_S^1$). To this end, it is critical to update $A_r(F)$ with signatures of new intrusions before these new intrusions are able to spread widely. The signature updating problem is rooted in the lack of efficient methods to update the signatures of new intrusions instantly (i.e., $A_r(F) \rightarrow A_i(F)$). With the zero-day worms/malware emerging nowadays, and it is indeed hard to update $A_r(F)$ with their signatures because of their short spreading cycle.

$\mathcal{P}_S^3$: Intrusion variation detection

Usually, we seek a one-to-one correspondence between signatures and intrusions in $A_r(F)$. However, it is possible that one intrusion will cause multiple signatures. Considering that $A_i(F)$ includes the signatures of all intrusions as well as all of their variations, the detection rate is degraded if $A_r(F)$ does not include intrusion variations. From another aspect, the intrusion variations can be considered as new intrusions for intrusion detection (thereafter, refer to $\mathcal{P}_S^1$).

CHAPTER 4. PROBLEM ANALYSIS

$\mathcal{P}_S^4$: Too many false alarms

Due to the inadequacy of the behavior model (i.e., *incompleteness and inaccuracy*), there exist three signature subsets affecting its detection performance: $AS_1(F)$, $AN_1(F)$ and $AS_2(F)$. It is obvious that the false alarm rate is completely dominated by these three sets given a constant (e.g., known) intrusion signature base. Also under the perfect scenario, the false alarm rate approaches zero for signature-based intrusion detection.

In summary, all problems are strongly related to the performance of SID, which depends on both quality factors, as well as the utilized features, which may cause the suspicious feature subspace to be non-empty, of the intrusion behavior model $A_r(F)$.

4.3.2 Anomaly-based Intrusion Detection

Definition 4.6 (anomaly-based intrusion detection) *It is an intrusion detection problem, that uses only a normal behavior model $N_r(F)$ derived from normal audit trails only. In the detection phase, the behavior violating $N_r(F)$ is regarded as an ‘anomaly’, i.e., the behavior does not match any signature in $N_r(F)$.*

Figure 4.3: The normal behavior model $N_r(F)$ in AID techniques, where (1) $a + b + c = 1$, (2) $|S_i(F)| = x * |N_i(F)|$, $|A_i(F)| = y * |N_i(F)|$, (3) $|NS_1(F)| + |NS_2(F)| = b * |N_r(F)|$, (4) $|NA_1(F)| = c * |N_r(F)|$.

In AID, $S_r(F) = \emptyset$, and $A_r(F) = \Omega(F) - N_r(F)$. In order to measure model inaccuracy and model incompleteness in $N_r(F)$, we introduce three additional parameters a , b , and c representing fractions. $a * |N_r(F)|$ is the number of signatures that are correctly categorized into $N_r(F)$, $b * |N_r(F)|$ is the number of signatures that are

CHAPTER 4. PROBLEM ANALYSIS

categorized incorrectly due to model incompleteness, and $c * |N_r(F)|$ is the number of signatures that are categorized incorrectly due to model inaccuracy.

From Figure 4.3, we can infer the four detection categories in Table 4.1 as,

$$\begin{aligned}
 |TP(F)| &= |A_i(F)| + |S_f(F)| * \alpha - |NA_1(F)| \\
 |FP(F)| &= |N_f(F)| + |S_f(F)| * (1 - \alpha) \\
 |FN(F)| &= (|NS_1(F)| + |NS_2(F)|) * \alpha + |NA_1(F)| \\
 &= |N_r(F)| * (b\alpha + c) \\
 |TN(F)| &= |N_r(F)| - (|NS_1(F)| + |NS_2(F)|) * (1 - \alpha) - |NA_1(F)| \\
 &= |N_r(F)| * (a + b\alpha)
 \end{aligned}$$

We can then express the detection performance as follows:

$$\begin{aligned}
 DR(F) &= 1 - \frac{|N_r(F)|}{|N_i(F)|} \times \frac{c + b\alpha}{y + x\alpha} \\
 FAR(F) &= \frac{|N_i(F)| * (x(1 - \alpha) + 1) - |N_r(F)|(1 - b\alpha - c)}{|N_i(F)| * (x + y + 1) - |N_r(F)|}
 \end{aligned}$$

In contrast to SID, the detection rate of AID will be improved with more (known and unknown) vulnerabilities (i.e., with the increase of y). Specifically, a good modeling technique, i.e., $b = c = 0$, will make the detection rate approach 100%, and it is promising for intrusion detection. Static analysis is such a good modeling technique for intrusion detection. Because of its special model building procedure from the source/binary code, static analysis (Wagner et al.[WD01]) has the potential to build a complete normal behavior model without inaccuracy. Thus, $N_f(F) = S_f(F) = A_f(F) = NS_1(F) = NA_1(F) = \emptyset$. $FAR(F) = \frac{x(1-\alpha)+b\alpha}{x+y}$ and $DR(F) = 1 - \frac{b\alpha}{y+x\alpha}$. However, some normal signatures may still be suspicious under some scenarios (i.e., $NS_2(F) \neq \emptyset$, $b \neq 0$ and $x \neq 0$) because we know nothing about any intrusion in static analysis. Thus, $FAR \neq 0$ and $DR \neq 100\%$. In this case, the existence of $NS_2(F)$ and $S_r(F)$ is caused by the improperly selected feature vector. To this end, a critical problem in static analysis is how to select the feature vector.

In addition, AID could also have an empty suspicious feature subspace to improve its efficiency if a proper feature vector is selected. Nonetheless, AID is not a panacea for intrusion detection because of the quality problem of $N_r(F)$ (i.e., $c \neq 0, b \neq 0$, and/or $x \neq 0$).

Similarly, the problems of AID can be analyzed as follows.

$\mathcal{P}_A^1$: Inability in identifying intrusions

Any intrusion outside $N_r(F)$ will trigger an ‘anomaly’ in AID techniques. However, there is no difference among these anomalies. To make it worse, some new normal behaviors outside $N_r(F)$ will also be detected as ‘anomalies’.

 $\mathcal{P}_A^2$: Higher false alarm rate

The reason for the higher false alarm rate lies in the implicit assumption that *anomalies detected are indications of intrusions*, which causes the future signature sets $N_f(F)$ and $S_f(F)$ to be anomalous. Basically, the model incompleteness makes most AID techniques lack distinguishability in the future resource behaviors, and it precipitates such implicit assumption. By increasing completeness, $N_f(F) \rightarrow \emptyset$, $S_f(F) \rightarrow \emptyset$, $NS_2(F) \rightarrow \emptyset$, and thus $FAR(F) \rightarrow 0$. That is to say, the false alarm rate is determined by the incompleteness of the behavior model $N_r(F)$.

 $\mathcal{P}_A^3$: Determining whether anomalies are caused by intrusions

In AID, all the behaviors falling into $\Omega(F) - N_r(F)$ are detected as anomalies. From Figure 4.3, the anomalies can be caused by *new normal behaviors* (i.e., model incompleteness), *intrusions*, and *new suspicious behaviors*. It is almost impossible to distinguish them in AID. The misclassified behaviors due to model incompleteness lead to $FP(F)$. The mechanism in [LD04b] is to make the signatures in $FP(F)$ flow into $TN(F)$, and thus to lower the false alarm rate.

 $\mathcal{P}_A^4$: Mimicry attacks

In principle, mimicry attacks [WS02] try to utilize the signatures in $NS_1(F)$ and $NS_2(F)$, which are ‘normal’ in the behavior model $N_r(F)$ but actually ‘suspicious’ in the *ideal* scenario. At the same time, if the attacker knows the model inaccuracy, the signatures in $NA_1(F)$ can also be utilized in mimicry attacks. Because of these three signature sets, mimicry attacks can lower the detection rate. In a converse viewpoint, mimicry attacks can thus achieve its objective(s) without being detected. Moreover, the success of mimicry attacks is based on the assumption that attackers are familiar with the detection mechanism and the normal behavior model.

$\mathcal{P}_A^5$: Concept drifting problem

Concept drifting will lead to more inaccurate labels in the training audit trails, and thus it will aggravate the inaccuracy problem in the normal behavior model $N_r(F)$. Therefore, the signature sets $NS_1(F)$, $NS_2(F)$ and $NA_1(F)$ are enlarged, and $N_f(F)$ is enlarged as well. It results from the reasonable assumption that the ideal signature sets $N_i(F)$, $S_i(F)$ and $A_i(F)$ keep constant with concept drifting. These affected signature sets will lower the detection rate and make the false alarm rate larger at the same time.

 $\mathcal{P}_A^6$: Intensive detection computation

To reduce model incompleteness, we should increase the number of signatures, or make the generalization on a signature as large as possible. Therefore, the size of the behavior models will be enlarged, and hence the detection computation will increase. To make matters worse, in AID techniques, all signatures should be checked before an anomaly is triggered.

In conclusion, the quality issues in the normal behavior model $N_r(F)$ lead to most problems in AID, and these problems affect its detection performance.

4.3.3 USAID: Unifying SID and AID

In our proposed technique USAID (*Chapter 6*) and specification-based techniques (Calvin et al.[KRL97], Sekar et al.[SGF⁺02]), all of known knowledge can be modeled and utilized for intrusion detection. That is, all three feature subspaces $N_r(F)$, $S_r(F)$ and $A_r(F)$ are known in advance. In the detection phase, if no signatures identify the current behavior in three behavior subspaces, it is an ‘anomaly’. In our following analysis, the notations in Figure 4.2 and 4.3 are applied. Using all the signature sets related to the quality of the behavior model, the detection categories can be inferred from the detection results.

$$\begin{aligned}
|TP(F)| &= |A_i(F)| - c * |N_r(F)| + |S_f(F)| * \alpha + f * |A_r(F)| * \alpha \\
|FP(F)| &= (f * |A_r(F)| + |S_f(F)|) * (1 - \alpha) + |N_f(F)| + g * |A_r(F)| \\
|FN(F)| &= c * |N_r(F)| + b * |N_r(F)| * \alpha \\
|TN(F)| &= a * |N_r(F)| + b * |N_r(F)| * (1 - \alpha)
\end{aligned}$$

The detection performance of USAID is expressed as follows:

$$\begin{aligned} DR(F) &= 1 - \frac{|N_r(F)|}{|N_i(F)|} \times \frac{c + b\alpha}{y + x\alpha} \\ FAR(F) &= \frac{|N_i(F)| * (x(1 - \alpha) + 1) - |N_r(F)|(1 - b\alpha - c)}{|N_i(F)| * (x + y + 1) - |N_r(F)|} \end{aligned}$$

It is clear that USAID will achieve the same detection performance as that of AID since both of them are affected by the quality factors in the behavior model, i.e., $NS_1(F)$, $NS_2(F)$, $NA_1(F)$. Like AID, if the behavior model is accurate and complete, $DR(F) = 100\%$ and $FAR(F) = 0$. Other than detecting anomalies, USAID can identify the intrusions in $A_r(F)$ as in SID. In summary, USAID can provide advantages of SID and AID while it is still limited by the quality issues of the behavior model.

4.4 Further Discussions

In [McH01], McHugh listed several open questions for intrusion detection, which remain unanswered till today. We will further discuss the first four basic questions under our framework while we left them as our future work:

- Soundness of Approach;
- Completeness of Approach;
- Timeliness of Approach;
- Choice of Metrics, Statistical Models, and Profiles.

Within our framework, it is possible to answer them practically according to the following steps. First, we can investigate the intrusion evolution in Internet, and then identify what are the significant (*or possible*) features for intrusion detection. Based on these features, it is possible to determine what extent of model generalization is needed to detect each intrusion instance (c.f., Chapter 5). After summarizing all such information, we can estimate the degree of model generalization to detect a set of intrusions (*or all past and current intrusions*).

Finally, we will generate artificial datasets with specific extent of model generalization to evaluate existing intrusion detection techniques. Based on the artificial datasets with adjustable quality, we can infer the practical answers for soundness and completeness questions. From the evaluation results, we can choose proper techniques/approaches and features to do intrusion detection. If an anomalous signature of an intrusion is detected before the damage is caused by the intrusion, the timeliness can be guaranteed as well.

4.5 Summary

In this chapter, we propose a theoretical basis for intrusion detection, in which the generic problems of SID and AID are analyzed quantitatively. In our analysis, a clear outline is drawn about the reasons why these problems exist. Two quality factors in the behavior model, namely, incompleteness and inaccuracy, as well as the utilized features, result in the generic problems for intrusion detection. Furthermore, the analysis lays out a more comprehensive outlook for intrusion detection, which has the potential to guide the future development of various solutions/techniques of intrusion detection. We also found that static analysis is promising because of its special model building method. An important point in static analysis is to select a proper feature vector. The detection rate will be degraded otherwise due to the possible existence of suspicious signatures in its behavior models.

Chapter 5

Model Generalization and Its Implications on Intrusion Detection

As analyzed in Chapter 4, there are two quality factors in behavior models, namely *model inaccuracy* and *model incompleteness*. In the following three paragraphs, we try to analyze what affects both quality factors, and thus affects the model quality, in intrusion detection.

What affects the model quality? For an intrusion detection system, the behavior models are generally built from known knowledge of a computing resource. The quality of the behavior models is thus determined by the known knowledge as well as the ‘*model building*’ technique. However, it is usually hard to guarantee the completeness of known knowledge (e.g., *with the concept drifting problem* $\mathcal{P}_A^5$) at least because of the lack of knowledge about future behaviors. The incompleteness of known knowledge leads to model incompleteness in the behavior models.

To make up for the incompleteness, most existing ‘*model building*’ techniques try to infer the status of an unknown behavior via *model generalization* (Esponda et al.[EFH02], Debar et al.[DDW00], Lee et al.[LS00]). The functionality of model generalization can be illustrated using the following example. If a packet size 1000-bytes is normal but 10-bytes is anomalous for a computing resource, then the size 900-bytes may be considered normal if we assume that a ‘new’ instance (e.g., 900-bytes) has the same status as its nearest instance (e.g., 1000-bytes as $|1000 - 900| < |900 - 10|$).

However, model generalization is not a panacea to solve the incompleteness problem since it infers part but not all of unknown behaviors. To make things worse, model generalization also leads to model inaccuracy (Figure 4.1). In other words, both quality factors are determined by model generalization in building behavior models. In order

to get the precise influence of the two quality factors, it is necessary to evaluate model generalization and its implications for intrusion detection.

Importance of the evaluation. There are at least three explicit applications of this evaluation result. First, it helps us to determine the deployment conditions for an existing intrusion detection technique by calculating the extent of generalization needed. For instance, stide, a typical host-based intrusion detector [FHSL96], is not efficient under network-based deployment [LD04a]. Secondly, it could guide the development of an adaptive intrusion detection technique by adjusting the extent of model generalization. In particular, the intrusion and application evolutions will help us to determine what extent of model generalization is needed in an ad hoc deployment. Lastly but most importantly, it could be applied for intrusion detection evaluation. We can generate artificial datasets with different extents of model generalization, and use them to identify generic detection capability of a technique. For above reasons, the evaluation for model generalization is a challenging but fundamental task for intrusion detection.

Most existing research have generally credited the overall efficiency of an intrusion detection technique to such model generalization, but have not perform any quantitative evaluation of the effect of model generalization. This is partially due to the difficulty of pinpointing the contribution of model generalization to the overall efficiency. Fortunately, our framework in Chapter 3 not only overcomes this problem, but also allows one to adjust the extent of model generalization in building (normal/intrusive) behavior models. Based on it , we design an evaluation framework for model generalization. Experiments are then performed to get the exact implications of model generalization on intrusion detection.

5.1 An Evaluation Framework

We first define the model generalization within the context of intrusion detection.

Definition 5.7 (Model Generalization) *Suppose that there exists a set of known behaviors associated with a resource. Within the context of intrusion detection, **model generalization** is an inference operation that tries to identify a new behavior associated with the same resource based on existing behavior instances.*

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

Model generalization can improve the detection rate by identifying more novel behaviors (e.g., normal behaviors) but may also degrade the detection performance by mis-identifying novel behaviors due to generalization errors (Valdes et al.[VS00]). This influence of model generalization on detection performance is generally determined by its underlying assumptions per se. In our evaluation, we first pinpoint three phases of our framework where we can use various assumptions to apply three levels of generalization (cf. section 3.2.5.1), and then evaluate these underlying assumptions respectively for model generalization. Note that we also include a level without any generalization in a sense that the behaviors in the training audit trails are represented in the behavior models precisely.

L1: Generalizing feature ranges.

The assumption is that there is a deviation in values of a feature among different runs of a behavior. In a *feature range collection strategy*, every feature range is generalized to cover more unknown parts of feature space other than its corresponding value(s) in the training audit trails.

L2: Generalizing behavior models.

The assumption is that, if two signatures are from the same feature subspace (i.e., with the same NSA label), their corresponding component feature ranges are equivalent. In this level of generalization, we can manipulate signatures in the behavior models, especially, two signatures with the same NSA label. Some signatures, which cannot be collected from the training audit trails, can be augmented into the behavior models. To some extent, the effect of L2 generalization is to ignore several features in building signatures.

L3: Generalizing signatures during detection.

The assumption is that an instance will be recognized by its nearest signature in a sense that the number of the values of the instance falling into the signature is maximal. With a *distance function*, the model generalization introduces some signatures that are not in the training audit trails but within a distance threshold from existing signatures.

These three levels of generalization is largely ranked from specialization (i.e., with less inference) to generalization (i.e., with more inference). In the last level, some errors may be caused even in detecting behavior instances in the training audit trails.

5.1.1 L0 Without Generalization

Suppose that for a feature F , there exists a series of values, $v_1, v_2, \dots, v_l$ in the training audit trails. In L0, every value v_i is regarded as a feature range with the upper and lower bounds equal to itself v_i . For this reason, the instances in the training audit trails are represented *precisely* by the signatures generated from these feature ranges. Note that, for the feature F , we have not inferred the NSA label of unknown subspace between any two values within $Dom(F)$.

5.1.2 L1 Model Generalization

To achieve L1 generalization for every feature, the unknown parts in the feature space should have the same NSA label as its neighboring values in our assumption. Inherent in this assumption is a concept of distance. Unfortunately, there is no distance concept within nominal features, so that L1 generalization is not applicable to them. In other words, we will only apply the L1 generalization on numerical (i.e., discrete and continuous) features, and regard every value of a nominal feature as a feature range. In addition, we use two more notations on a feature range R_F^i : $Upp(R_F^i)$ is its upper bound and $Low(R_F^i)$ is its lower bound. With respect to a feature value v_i , an initial feature range R_F^i is formed with $Upp(R_F^i) = Low(R_F^i) = v_i$.

Algorithm 1 L1 Model generalization for a discrete or continuous feature F .

Require: (1) Initial feature ranges R_F^1, R_F^2 , and R_F^l . (2) Value precision ε (ε_d for discrete features and ε_c for continuous features as in Chapter 3).

```

1: /* lines 2-7 constitute the splitting step */
2: for  $i = 1$  to  $l - 1$  do
3:   determine a splitting border  $S$  within  $(Upp(R_F^i), Low(R_F^{i+1}))$ ;
4:   split  $(Upp(R_F^i), Low(R_F^{i+1}))$  into  $(Upp(R_F^i), S]$  and  $(S, Low(R_F^{i+1}))$ ;
5:   extend  $R_F^i = (Low(R_F^i), S]$  without changing its NSA label;
6:   extend  $R_F^{i+1} = (S, Upp(R_F^{i+1}))$  without changing its NSA label;
7: end for
8: /* the following merging step is optional */
9:  $i=1$ ;
10: while  $i < l$  do
11:   if  $Low(R_F^{i+1}) - Upp(R_F^i) \leq \varepsilon$ , and  $L(R_F^i) = L(R_F^{i+1})$  then
12:     merge  $R_F^{i+1}$  into  $R_F^i$ ;
13:     delete  $R_F^{i+1}$ ; {the index of every following feature range is decreased by 1}
14:      $l = l - 1$ ;
15:   else
16:      $i = i + 1$ ;
17:   end if
18: end while

```

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

L1 model generalization is given in Algorithm 1 (see Section 3.6.2 for the precision values ε_d and ε_c). In this generalization, an important step is to split the unknown subspace $(v_i, v_{i+1}) = (Upp(R_F^i), Low(R_F^{i+1}))$ ($i + 1 \leq l$), and allocate the two parts to existing neighboring ranges R_F^i and R_F^{i+1} . We call this step as the Unknown Subspace Splitting Step. We use several splitting strategies and evaluate them in our framework. They are: (1) no splitting (2) equal splitting, (3) frequency-based splitting, (4) intrusion-specific splitting. To make the generalization configurable, the Range Merging Step (in lines 9-18) is optional in Algorithm 1 after the splitting step (in lines 2-7) while it is also contributing to L1 generalization.

- (1) **L1.1: No splitting.** We do not conduct any splitting in L1 generalization. If we do not conduct the merging step either, the L1.1 generalization actually becomes same as L0, i.e., no generalization per se.
- (2) **L1.2: Splitting it equally.** The unknown interval between v_i and v_{i+1} is split at the midpoint at $S = \frac{v_i + v_{i+1}}{2}$. That is, $(v_i, S]$ is assigned the same NSA label as v_i , and $(S, v_{i+1}]$ is assigned the same NSA label as v_{i+1} .
- (3) **L1.3: Splitting it based on frequency of neighboring values.** Let the frequency of v_i in the training audit trails be $Freq_{v_i}$. Then, the splitting point is determined as $S = v_i + (v_{i+1} - v_i) * \frac{freq_{v_i}}{freq_{v_i} + freq_{v_{i+1}}}$. $(v_i, S]$ is assigned the NSA label of v_i , and $(S, v_{i+1}]$ is assigned the NSA label $L(v_{i+1})$.
- (4) **L1.4: Intrusion specific splitting [VS00].** Given a predefined generalization parameter G_{in} for intrusions. For a pair of neighboring values v_i and v_{i+1} , if $L(v_i) = 'N'$ and $L(v_{i+1}) = 'A'$, $S = v_{i+1} - G_{in}$. If $L(v_i) = 'A'$ and $L(v_{i+1}) = 'N'$, $S = v_i + G_{in}$. Otherwise, $S = \frac{v_i + v_{i+1}}{2}$. $(v_i, S]$ is assigned the same NSA label as v_i , and $(S, v_{i+1}]$ is assigned the same NSA label as v_{i+1} .

As in Algorithm 1, we will also evaluate the generalization in the range merging step.

After the L1 model generalization, the feature ranges for every feature are collected, and the behavior models $\Omega(F_{1..n})$ are constructed. In the detection phase, every test instance is first formalized as a test signature Sig_t by replacing every value with its corresponding feature range. Then, we evaluate whether Sig_t matches any signature in $\Omega(F_{1..n})$. If it matches a signature, it will be identified by that signature. Otherwise, Sig_t will further be evaluated by L2/L3 model generalization evaluation processes.

5.1.3 L2 Model Generalization

After the L1 model generalization, all the (nominal, discrete, and/or continuous) features are uniformly represented by a series of feature ranges. In L2 model generalization, we will utilize the relations between feature ranges rather than values.

To achieve L2 generalization, the distance of two signatures will be utilized to measure the extent of generalization. To this end, let us first define a distance function of two signatures in the behavior models.

5.1.3.1 Signature Distance

Let $R(Sig_1, F_i)$ denote the feature range of the feature F_i in a signature Sig_1 . For any two signatures, Sig_1 and Sig_2 , their distance is defined as follows.

$$D(Sig_1, Sig_2) = \sum_{i=1}^n \delta(Sig_1, Sig_2, F_i)$$

where,

$$\delta(Sig_1, Sig_2, F_i) = \begin{cases} 0, & \text{if } R(Sig_1, F_i) = R(Sig_2, F_i); \\ 1, & \text{otherwise.} \end{cases}$$

5.1.3.2 Evaluating L2 Model Generalization

The main objective of L2 generalization is to recognize the signature Sig_t , which has not been identified by $\Omega(F_{1...n})$. In our evaluation, L2 generalization is achieved by the following two generalization operations.

- **L2.1: Grouping feature ranges.** If several feature ranges of a feature are interchangeable in $\Omega(F_{1...n})$ without loss of signature distinguishability, they will be combined into a group. A signature $Sig_1 (\in \Omega(F_{1...n}))$ is equivalent to Sig_t if for every feature F_i , either $\delta(Sig_t, Sig_1, F_i) = 0$ or $\delta(Sig_t, Sig_1, F_i) = 1$ but $R(Sig_t, F_i)$ and $R(Sig_1, F_i)$ fall into the same group.
- **L2.2: Mutating feature ranges.** For a feature, its feature range in a signature can be mutated to any of its other feature ranges. From another aspect, all the feature ranges of the feature form one group and thus it is no use in our detection.

Example Signatures. We use the example signatures in Table 5.1 (*which is the same example in Chapter 3*) to illustrate the L2 model generalization.

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

Index	Signature [(R_{F_1} , R_{F_2} , R_{F_3}), NSA Label, Status List]				
Sig_1	[(TCP,	[1,2],	[0.01,0.05]),	‘N’,	‘Normal’]
Sig_2	[(ICMP,	[1,2],	[0.01,0.05]),	‘N’,	‘Normal’]
Sig_3	[(TCP,	[3,5],	(0.05,0.09)],	‘A’,	‘ I_1 ’]
Sig_4	[(UDP,	[3,5],	(0.05,0.09)],	‘A’,	‘ I_2 ’]
Sig_5	[(NETBIOS,	[6,6],	(0.09,0.12)],	‘S’,	‘Normal, I_3 ’]
Sig_6	[(UDP,	[7,7],	[0.01,0.05]),	‘N’,	‘Normal’]
Sig_7	[(UDP,	[8,8],	(0.12,0.14)],	‘A’,	‘ I_2, I_4 ’]

Table 5.1: The behavior signatures built from the instances in Table 3.2, where ‘I’ denotes the status ‘intrusion’, and ‘Normal’ represents normal behaviors. We have used the L1.2 splitting strategy and the range merging step in building feature ranges. We only list signatures here to illustrate L2 and L3 model generalization.

5.1.3.3 Grouping Feature Ranges

In $\Omega(F_{1...n})$, if two feature ranges of a feature F_i are replaceable with each other and the replacement will not lead to any loss of signature distinguishability, their significance is equal to each other. These feature ranges can be grouped without loss of the distinguishability. For example, in Table 5.1, two feature ranges ‘TCP’ and ‘ICMP’ of F_1 is in a group. As a special case, a feature range can form a group by itself.

Using this grouping method, a series of groups is formed for F_i , $G_{F_i} = \{G_{F_i}^1, G_{F_i}^2, \dots\}$ such that for any feature range $R_{F_i}^j$, there is a group $G_{F_i}^k$, $R_{F_i}^j \in G_{F_i}^k$. Finally, we achieve a **grouping scheme** for all features in the feature vector: $G_{FV} = \langle G_{F_1}, G_{F_2}, \dots, G_{F_n} \rangle$.

Let us talk more about the relations among signatures with respect to a grouping scheme. For two signatures Sig_1 and Sig_2 in $\Omega(F_{1...n})$, they are **equivalent** to each other with respect to G_{FV} based on the following rule.

$$Sig_1 \stackrel{G_{FV}}{\equiv} Sig_2 \Leftrightarrow \forall i(\delta(Sig_1, Sig_2, F_i) = 1 \vee (\delta(Sig_1, Sig_2, F_i) = 0 \wedge \exists j\{R(Sig_1, F_i), R(Sig_2, F_i)\} \subset G_{F_i}^j))$$

For any two **equivalent** signatures, they are **compatible** if they have the same NSA label. Otherwise, they are **conflict** to each other in the behavior models.

The behavior models can be generalized (i.e., enlarged) by grouping feature ranges. For example, for signatures “[$a, 1, E$]” and “[$b, 2, F$]”, if ‘a’ and ‘b’ are grouped, the behavior models can be enlarged by two additional signatures “[$b, 1, E$]” and “[$a, 2, F$]”. In Table 5.1, Sig_2 and Sig_3 are such signatures. Fundamentally, the *crossover* operation is allowed between signatures by interchanging the feature ranges in a group [Gol89].

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

Lemma 5.1 *With respect to a grouping scheme G_{FV} , any two equivalent signatures in the behavior models are **compatible**.*

Lemma 5.2 *With respect to a grouping scheme G_{FV} , if a test signature Sig_t is equivalent to a signature Sig_j in the behavior models, then $L(Sig_t) = L(Sig_j)$.*

In order to limit the diversity in a grouping scheme, we also define a parameter np_g for the number of the grouping points. Specifically, for two equivalent signatures Sig_t and Sig_j in the grouping scheme, they share at least $n - np_g$ identical feature ranges. The larger the parameter np_g is, the more diverse the group operation is. Given np_g and $\Omega(F_{1...n})$, the test instance Sig_t is evaluated as in Algorithm 2.

The identification of Sig_t is conducted in a special way. Instead of mining the whole scheme, we only want to determine whether Sig_t is equivalent to Sig_j ($\in \Omega(F_{1...n})$) without violating the grouping scheme. Specifically, for every signature Sig_j with $D(Sig_t, Sig_j) \leq np_g$, we collect a feature range pair P for every feature F_l with $\delta(Sig_t, Sig_j, F_l) = 0$. A test on P is performed to determine whether there is conflicting signature in $\Omega(F_{1...n})$ with respect to P . The lack of confliction will prove the existence of P in a grouping scheme G_{FV} . If all such range pairs exist in G_{FV} , the signature is identified.

Algorithm 2 Evaluating a test signature via grouping w.r.t. $\Omega(F_{1...n})$.

Require: (1) $\Omega(F_{1...n})$; (2) a test signature Sig_t ; and (3) the number of grouping points np_g .

```

1: Initialization,  $StatusList = \emptyset$ 
2: for every signature  $Sig_j \in \Omega(F_{1...n})$  do
3:   if  $D(Sig_t, Sig_j) \leq np_g$  then
4:     /* if  $R(Sig_j, F_i) \neq R(Sig_t, F_i)$ ,  $P = \{R(Sig_j, F_i), R(Sig_t, F_i)\}$  */
5:     enumerate all feature range pairs  $P_1, P_2, \dots, P_k$  ( $k \leq np_g$ );
6:     test whether there are conflicting signatures in  $\Omega(F_{1...n})$  w.r.t.  $P_1, P_2, \dots, P_k$ ;
7:     if there are no confliction on  $P_1, P_2, \dots, P_k$  then
8:       accumulate the status(es) of  $Sig_j$  into  $StatusList$ ; /* Lemma 5.2 */
9:     end if
10:  end if
11: end for
12: if  $StatusList \neq \emptyset$  then
13:   give the detection results based on  $StatusList$ ;
14: else
15:   output as an anomaly;
16: end if
```

If the output is an anomaly, we will further evaluate Sig_t whether it can be evolved by the following two generalization operations (i.e., mutation and nearest signatures).

5.1.3.4 Mutating Feature Ranges

Ignoring some features will cause a signature to identify more behaviors. For example, if all three features are utilized, the instance ‘ $F_1=\text{UDP}, F_2=8, F_3=0.51$ ’ will not be identified by any signature in Table 5.1. But if the third feature F_3 is ignored, the signature Sig_7 will identify the instance as an ‘attack’. Essentially, ignoring features is equal to the mutation operation in Genetic Algorithms[Gol89]. The mutation should not cause any confliction among all the signatures either.

If the test signature Sig_t cannot be recognized after grouping feature ranges, the feature range mutation is allowed to identify Sig_t . In effect, the identification is conducted by ignoring some inconsistent feature ranges between the test signature and the matched signature. One condition of a successful mutation is that it should not lead to any contradiction in the existing signatures. For example, if we let the first two features mutate within this generalization, the signatures “ $\langle a, b, c, d \rangle$ ” in $N(F_{1...4})$ and “ $\langle x, y, c, d \rangle$ ” in $A(F_{1...4})$ will contradict to each other.

Algorithm 3 Evaluating a test signature via mutation w.r.t. $\Omega(F_{1...n})$.

Require: (1) $\Omega(F_{1...n})$; (2) a test signature Sig_t ; and (3) the number of mutation points np_m .

```

1: Initialization,  $StatusList = \emptyset$ 
2: for every signature  $Sig_j \in \Omega(F_{1...n})$  do
3:   if  $D(Sig_t, Sig_j) \leq np_m$  then
4:     /* if  $R(Sig_j, F_i) \neq R(Sig_t, F_i)$ , the feature  $F_i$  will be mutated in our evaluation */
5:     enumerate all mutated features  $F_{m_1}, F_{m_2}, \dots, F_{m_k}$  ( $k \leq np_m$ );
6:     test whether there are conflicting signatures in  $\Omega(F_{1...n})$  after mutating  $F_{m_1}, F_{m_2}, \dots, F_{m_k}$ ;
7:     if there are no confliction then
8:       accumulate the status(es) of  $Sig_j$  into  $StatusList$ ;
9:     end if
10:  end if
11: end for
12: if  $StatusList \neq \emptyset$  then
13:   give the detection results based on  $StatusList$ ;
14: else
15:   output as an anomaly;
16: end if
```

Similar to the above grouping operation, we use a mutation point number np_m to measure the diversity of the mutation process. In the detection phase, given np_m and $\Omega(F_{1...n})$, the test signature Sig_t is evaluated as in Algorithm 3. While every signature can be mutated to extend behavior models, the identification is performed differently. For every signature Sig_j with $D(Sig_t, Sig_j) \leq ng_g$, all features with different feature ranges are collected, $F_{m_1}, F_{m_2}, \dots, F_{m_k}$ ($k \leq np_m$). For any $1 \leq l \leq k$, $R(Sig_1, F_{m_l}) \neq$

$R(Sig_t, F_{m_l}))$. We then test whether there is confliction in $\Omega(F_{1...n})$ after mutating these features. The lack of confliction will indicate the identification by Sig_j .

5.1.4 L3 Model Generalization

If the test signature Sig_t cannot be identified by L1 and L2 generalization, it will be identified by the signature(s) with the minimum signature distance to it.

5.1.4.1 Nearest Signatures

In the L3 generalization process, we assume that the test signature has the same NSA label as its nearest signature(s). To achieve it, we first calculate the minimum distance to all signatures in $\Omega_{F_{1...n}}$,

$$D_{min}(Sig_t, \Omega(F_{1...n})) = \min_{Sig_i \in \Omega(F_{1...n})} D(Sig_i, Sig_t)$$

Then, we accumulate the statuses of all signatures with a distance $D_{min}(Sig_t, \Omega(F_{1...n}))$ to the test signature. The detection result of the test signature Sig_t is given based on the accumulated statuses. For example, using the signatures in Table 5.1, the instance ' $F_1=TCP, F_2=7, F_3=0.04$ ' is detected as 'Normal' because $D_{min}(Sig_t, \Omega(F_{1...3})) = 1$ and two signatures Sig_1 and Sig_6 have a distance 1 to the test signature.

5.1.5 Measuring the Detection Performance

There are two main objectives for intrusion detection: **(1)** to detect the intrusions correctly, and **(2)** to identify the behaviors correctly. To reflect these two objectives, we assign a cost scheme as in Table 5.2 to quantify the detection performance. If the behavior is identified correctly, the cost is 0. Otherwise, some penalty is assigned. In particular, the detection of an intrusion as an anomaly is less useful than intrusion identification. Lastly, the average cost of every test instance is utilized to quantify the detection performance to evaluate the generalization within the behavior models (Lee et al.[LMS00]).

Suppose that there are T test instances. Based on the detection results, several statistics are further defined in Table 5.2. The number of false positives is $\#_{NA}$, and for false negatives, it is $\#_{IN}$. The average cost of every test instance is defined as:

$$cost = \frac{\#_{NA} \times 3 + \#_{IN} \times 3 + \#_{IA} \times 1}{T} \quad (\text{Eq. 5.1})$$

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

Index	Notation	Original Class	Identical Signature	Detection Result	Cost
1	$\#_{NN}$	normal	Y	normal	0
2	$\#_{NA}$	normal	N	anomaly	3
3	$\#_{II}$	intrusion	Y	original intrusion	0
4	$\#_{IA}$	intrusion	N	anomaly	1
5	$\#_{IN}$	intrusion	Y	normal	3

Table 5.2: Detection results and their costs.

The average cost in absence of any model generalization gives the reference baseline, $cost_{base}$, of the detection performance as there are no model errors. The usefulness of model generalization is reflected in the relation between its average cost and $cost_{base}$. If $cost > cost_{base}$, the performance has been degraded by such generalization. Otherwise, the model generalization is useful for intrusion detection.

5.2 Experiments: A Case Study

We have evaluated the generalization using the datasets in Chapter 3. However, the evaluation procedure is very computation-intensive so that we can finish the evaluation on the whole audit trails. In order to keep the computation within reasonable limits, we give samples on the training and test audit trails. 10000 instances are sampled from the total 4898431 training instances and 500 from the 311029 test instances. To make it convincing, we give three pairs of such training and test samples.

5.2.1 Without Model Generalization

Sample Pair	Normal	Intrusion	$\#_{NN}$	$\#_{NA}$	$\#_{II}$	$\#_{IA}$	$\#_{IN}$	$cost$
Pair 1	103	397	0	103	203	193	1	1.01
Pair 2	91	409	0	91	216	193	0	0.932
Pair 3	108	392	5	103	193	198	1	1.02

Table 5.3: L0: without model generalization. The 2nd and 3rd columns give the numbers of normal and intrusive instances in every sample pair respectively.

If there is no generalization, the instances in the training audit trails can be represented precisely. An instance in the test sample is identified only if it is identical to any instance in the training sample. Table 5.3 lists the detection results when there is no generalization, and they are acted as the baseline $cost_{base}$ in the following evaluations.

Among the detection results, the detection results on normal and attack instances have significant difference. More than half of intrusive instances are identified correctly (denoted by $\#_{II}$), but, in comparison, almost all normal instances are detected

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

incorrectly. To some extent, the fact indicates that the normal behaviors are of great variety, and more generalization is needed to infer their statuses.

5.2.2 Evaluating L1 Model Generalization

Table 5.4 gives the evaluation results on the L1 model generalization. For L1.4 generalization, we set $G_{in} \in \{0, 1, 2, 3, 4, 5, 10, 20\}$.

(Pair 1) Normal:Intrusion=103:397

Settings		With the range combining step						Without the range combining step					
L1	G_{in}	# NN	# NA	# II	# IA	# IN	$cost$	# NN	# NA	# II	# IA	# IN	$cost$
L1.1	-	6	97	241	155	1	0.898	0	103	203	193	1	1.01
L1.2	-	35	68	281	114	2	0.648	6	97	279	117	1	0.822
L1.3	-	35	68	280	115	2	0.65	6	97	278	118	1	0.824
L1.4	0	35	68	280	115	2	0.65	6	97	278	118	1	0.824
L1.4	1	35	68	280	115	2	0.65	6	97	278	118	1	0.824
L1.4	2	35	68	281	114	2	0.648	6	97	278	118	1	0.824
L1.4	3	35	68	280	115	2	0.65	6	97	279	117	1	0.822
L1.4	4	35	68	281	114	2	0.648	6	97	279	117	1	0.822
L1.4	5	35	68	281	114	2	0.648	6	97	278	118	1	0.824
L1.4	10	35	68	280	115	2	0.65	6	97	279	117	1	0.822
L1.4	20	35	68	281	114	2	0.648	6	97	278	118	1	0.824

(Pair 2) Normal:Intrusion=91:409

Settings		With the range combining step						Without the range combining step					
L1	G_{in}	# NN	# NA	# II	# IA	# IN	$cost$	# NN	# NA	# II	# IA	# IN	$cost$
L1.1	-	3	88	263	144	2	0.828	0	91	216	193	0	0.932
L1.2	-	39	52	294	113	2	0.55	7	84	294	113	2	0.742
L1.3	-	39	52	292	115	2	0.554	7	84	294	113	2	0.742
L1.4	0	39	52	290	117	2	0.558	7	84	290	117	2	0.75
L1.4	1	39	52	291	116	2	0.556	7	84	291	116	2	0.748
L1.4	2	39	52	291	116	2	0.556	7	84	291	116	2	0.748
L1.4	3	39	52	292	115	2	0.554	7	84	292	115	2	0.746
L1.4	4	39	52	293	114	2	0.552	7	84	293	114	2	0.744
L1.4	5	39	52	294	113	2	0.55	7	84	294	113	2	0.742
L1.4	10	39	52	294	113	2	0.55	7	84	294	113	2	0.742
L1.4	20	39	52	294	113	2	0.55	7	84	294	113	2	0.742

(Pair 3) Normal:Intrusion=108:392

Settings		With the range combining step						Without the range combining step					
L1	G_{in}	# NN	# NA	# II	# IA	# IN	$cost$	# NN	# NA	# II	# IA	# IN	$cost$
L1.1	-	9	99	234	154	4	0.926	5	103	193	198	1	1.02
L1.2	-	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.3	-	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.4	0	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.4	1	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.4	2	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.4	3	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.4	4	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.4	5	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.4	10	45	63	273	115	4	0.632	10	98	273	115	4	0.842
L1.4	20	45	63	273	115	4	0.632	10	98	273	115	4	0.842

Table 5.4: L1: model generalization (L1.1~4)

5.2.2.1 The Utility of the Unknown Subspace Splitting Step

Note that L1.1, which is the generalization without the range merging step, is actually similar to L0, which has no generalization at all. Comparing the detection results in Table 5.4 and Table 5.3, it is apparent that L1 model generalization is useful for identifying (normal and intrusive) behaviors in all three test samples. The generalization led to by the unknown subspace splitting step in Algorithm 1 is useful to identify more instances, and significantly so for intrusive behaviors.

5.2.2.2 The Utility of the Range Merging Step

The range merging step has contributed much to the performance enhancement by identifying more normal behaviors but it has little effect on the identification ability for intrusive behaviors. This phenomenon, along with the insignificant influence from G_{in} , supports our above assertion that the normal behaviors are of great variety but intrusive behaviors are not so diverse.

5.2.2.3 The Difference Between L1.2/3/4

The new false negatives caused by L1 generalization is negligible in all three sample pairs (with 1, 2 and 3 additional false negatives respectively). The different splitting techniques of L1.2/3/4 have little effect on the detection results.

In summary, L1 generalization is useful for intrusion detection but the detection results are not sensitive to the splitting strategies. It is worth noting that, because they are less diverse than normal behaviors, most intrusions will be identified even if there is no generalization. In comparison, for detecting normal instances, more model generalization must be performed. This observation is consistent with the results of Valdes et al.[VS00]. For this reason, along with the negligible difference between L1.2/3/4 generalization, we select L1.4 with $G_{in} = 0$ in the following experiments.

5.2.3 Evaluating L2 Model Generalization

Figures 5.1 and 5.2 list the evaluation results highlighting the influence of grouping and mutation operations on intrusion detection. In both figures, we only illustrate $\#_{NA}$, $\#_{IA}$ and $\#_{IN}$. The numbers of $\#_{NN}$ and $\#_{II}$ can be deduced from the figures since the total of normal and intrusive behaviors remains constant.

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

5.2.3.1 L2.1 Grouping Generalization.

5.1.a: Pair 1 (Normal:Intrusion=103:397).

5.1.b: Pair 2 (Normal:Intrusion=91:409).

5.1.c: Pair 3 (Normal:Intrusion=108:392).

5.1.d: Overall Performance w.r.t. nGroup

Figure 5.1: L2: model generalization - grouping (nMutate=0).

First, we evaluate the influence of the grouping operation on intrusion detection with $nMutate=0$. It is obvious that the grouping operation enhances the performance by identifying more normal and intrusive behaviors. Figure 5.1 shows that the detection performance on the three samples has the same characteristics. Specifically, the overall detection performance improves because of a reduction in the detection cost. With the increase of $nGroup$, $\#_{NN}$ and $\#_{II}$ increase while $\#_{NA}$ and $\#_{IA}$ decreases, all of which are desirable. One negative aspect of grouping generalization is the increase of $\#_{IN}$ with the slight increase of $nGroup$.

Overall, the generalization from the grouping mechanism is useful for intrusion detection even though it will lead to a few more false negatives. Considering the trade-off between gains and loss for detection performance, we choose $nGroup = 3$ in the following experiments.

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

5.2.3.2 L2.2 Mutation Generalization.

5.2.a: Pair 1 (Normal:Intrusion=103:397).

5.2.b: Pair 2 (Normal:Intrusion=91:409).

5.2.c: Pair 3 (Normal:Intrusion=108:392).

5.2.d: Overall Performance w.r.t. nMutate

Figure 5.2: L2: model generalization - mutation (nGroup=3).

In Figure 5.2, the improvements caused by L2.2 mutation generalization is not that significant as L2.1 or L1 generalization. The decreased extent of false positives, $\#_{NA}$, is neutralized by the increased extent of false negatives, $\#_{IN}$. This fact is also reflected by the overall detection cost in subfigure 5.2.d, which is reduced only by a very small, even negligible, extent. The mutation generalization will further worsen the negative aspects in grouping generalization.

In summary, the L2.2 mutation generalization is useful but it is not that significant. We select $nMutate = 5$ in evaluating L3 model generalization.

5.2.4 Evaluating L3 Model Generalization

In the L3 generalization evaluation, $nGroup = 3$ and $nMutate = 5$ (Table 5.5). In the sample pair 1, all the normal behaviors are identified correctly, and most intrusions are also identified correctly (352/397). In pair 1/2/3, most normal behaviors can be

identified correctly with fewer false positives (i.e., $\#_{NA}$) after the model generalization (from L1 to L3). In contrast, even though more intrusive behaviors are identified correctly as well with more generalization, the false negatives (i.e., $\#_{IN}$) increase to a large extent (in comparison with Table 5.3).

Sample Pair	Normal	Intrusion	$\#_{NN}$	$\#_{NA}$	$\#_{II}$	$\#_{IA}$	$\#_{IN}$	<i>cost</i>
Pair 1	103	397	103	0	352	4	41	0.254
Pair 2	91	409	89	2	375	9	25	0.18
Pair 3	108	392	105	3	353	2	37	0.244

Table 5.5: L3: model generalization (nGroup=3, nMutate=5)

All in all, model generalization is necessary in identifying more behaviors correctly. More specifically, as a result of model generalization, both true positives and true negatives increase significantly. Its negative aspect is that it will slightly increase the false negatives in the detection results while the false positives decrease. In other words, it is hard to reduce both false negatives and false positives at the same time.

5.2.5 The Implications of Model Generalization

The significance of model generalization for intrusion detection is summarized in Table 5.6. It is obvious that different levels of model generalization lead to different implications on intrusion detection. For example, the abstraction attack scenario in STAT family sensors (Kemmerer and Vigna[VK99]) is achieved by L2 & L3 generalization. Considering that STAT is successful in detecting new intrusions, our experimental results are consistent with this fact. In addition, even though true positives decrease with model generalization, the intrusion identification capability increases a lot. It is shown by the increase of $\#_{II}$ and the decrease of $\#_{IA}$ in Tables 5.4 and 5.5.

Mapping back to our analysis in Chapter 4, model generalization has alleviated the model incompleteness in the cost of deteriorating the model inaccuracy. There is a tradeoff between pains and gains. L1/L2.1 model generalization makes the behavior models more complete with a little increase on inaccuracy while, in L2.2/L3 model generalization, the benefits by increasing completeness have been offset by the increase of the pain of inaccuracy.

CHAPTER 5. MODEL GENERALIZATION AND ITS IMPLICATIONS ON INTRUSION DETECTION

Levels	TP	FP	TN	FN	Significance
L0, L1.1	-	-	-	-	they act as an evaluation baseline to indicate whether model generalization is necessary for intrusion detection. We also found that most intrusions are identified even without generalization.
L1.2/3/4	-	↓	↑	-	They improve the detection performance in our case study, significantly for intrusive behaviors. Most importantly, they lead to only a few more false negatives. Their difference are negligible.
Range Merging	-	↓	↑	-	It is very useful to infer the statuses for normal behaviors, but it contributes less in identifying intrusive behaviors. Another good point is that it does not lead to more false negatives.
L2.1	↓	↓	↑	↑	The identification capability is significantly lifted with decreasing anomalies. However, there is an optimal value for the number of grouping points, which should be determined in advance.
L2.2 /L3	↓	↓	↑	↑	The identification capability is slightly lifted with decreasing anomalies. But the increase extent of false negatives is so large that we should neglect the increase of identification capabilities.

Table 5.6: The significance of different levels of model generalization. The symbol ‘↓’ represents ‘**decrease**’ and the symbol ‘↑’ represents ‘**increase**’. ‘-’ denotes that it will not affect the parameter. TP: true positive ($\#_{II} + \#_{IA}$), FP: false positive ($\#_{NA}$), TN: true negative ($\#_{NN}$), FN: false negative ($\#_{IN}$).

5.3 Summary

In this chapter, we designed a framework to evaluate the usefulness of model generalization on intrusion detection, in which the model generalization was divided into several levels and their underlying assumptions are identified. In a case study, we found that L1 model generalization is generally useful to identify more ‘novel’ behaviors, especially for normal behaviors. L2.1 generalization will benefit intrusion detection by significantly improving the identification capability even though false negatives will increase slightly. The gains and loss by applying L2.2 and L3 generalization should be considered seriously under different application scenarios.

Even though the proposed evaluation framework for model generalization is able to generally applied to most scenarios, our conclusions are only based on a typical dataset for intrusion detection. One of our further work is to apply this framework to evaluate the utility of model generalization on other datasets.

Chapter 6

Unifying Signature-based and Anomaly-based Intrusion Detection

Inspired by theoretical and experimental analysis results, we propose a new intrusion detection technique, **Unified Signature-based and Anomaly-based Intrusion Detection** (USAID). Unlike SID and AID techniques that utilize partial knowledge of a resource, USAID makes effective use of all the available knowledge about the intrusive and normal behaviors in the historical audit trails, and all the existing features are applied to detect (normal/intrusive) behaviors in the detection phase.

Figure 6.1: A general architecture for USAID.

The architecture of USAID consists of three modules as illustrated in Figure 6.1. The first module extracts the three feature subspaces for every feature. The signature building module is to construct a signature base including normal, suspicious and attack signatures, named as *NSA signature base*. The last module incorporates detection mechanisms using the signatures in the NSA signature base. To make USAID effective, `labeled audit trails` with intrusive and normal behaviors are needed.

6.1 Feature Subspaces Extraction

In the first module, there are three steps.

Step 1: Feature Value Collection. In the labeled training audit trails, all values are collected for every feature. The status (*‘normal’ and/or ‘attack’ depending on whether it occurs in normal or attack audit trails*) of every value is also collected and accumulated in its status list. Based on its status list, each value is assigned an NSA label. This step is applied to nominal, discrete and continuous features, while the following two steps are only applicable to discrete and continuous features.

Step 2: Feature Value Clustering. The objective of this step is to form initial feature ranges for every feature by clustering the neighboring values. For a discrete feature, two values x_1 and x_2 are *neighboring* if $|x_1 - x_2| = \varepsilon_d^1$. For a continuous feature, two values x_1 and x_2 are *neighboring* if their distance is smaller or equal to its value precision ε_c , i.e., $|x_1 - x_2| \leq \varepsilon_c$. If several neighboring values have the same NSA label, they are combined to form an *initial feature range*. As a special case, if, for a value, its neighbors have different NSA labels from itself, it forms an initial feature range whose upper and lower bounds are both equal to the value itself. Every initial feature range thus formed inherits the NSA label of the values falling within it.

Step 3: Feature Range Generalization. However, under most scenarios, the initial feature ranges of a feature will not cover all of its feature space. Any feature subspace outside the labelled feature ranges is named as an *uncovered space*. Two feature ranges are called *adjacent* if there is no other labelled feature range between them. The uncovered space between any two adjacent feature ranges is treated as follows: if the two adjacent feature ranges have the same NSA label, a new larger feature range is formed that includes the two feature ranges as well as the intervening uncovered space. This new range is assigned the NSA label of the ranges at the two edges. If the NSA labels of the two adjacent ranges are different, the uncovered space is divided *equally* (cf. Equal splitting in Section 5.1) and allocated to these two defined feature ranges. The NSA labels of the initial feature ranges will be inherited by the newly extended or combined feature ranges. As a result, finally, all the *known* feature space of every feature will be covered by well-defined feature ranges.

¹In USAID, $\varepsilon_d = 1$. If the distance between two discrete feature values is larger than 1, there will be an uncovered space with size $|x_1 - x_2 - 1|$ that will be generalized in step 3.

CHAPTER 6. UNIFYING SIGNATURE-BASED AND ANOMALY-BASED INTRUSION DETECTION

More details on these three steps can be found in the L0/L1 generalization of the Section 5.1. Table 6.1 is an example of feature ranges generated in the experiments for the discrete feature “hot” (“*number of ‘hot’ indicators* [Elk99]”). In this table, for example, for the feature range ‘[27,29]’ with lower bound 27 and upper bound 29, there are 5 instances labelled as ‘normal’ and 274 instances labelled as ‘warezclient’ in the training audit trails, and thus, its NSA label is ‘S’.

INDEX	NSA	RANGE	INTRUSIONS AND THEIR FREQUENCY
0	S	[0,7]	“normal:971107”, “perl:3”, . . . , “buffer_overflow:30”, “0:1”, “phf:4”
1	N	[8,9]	“normal:14”
...	...	...	...
8	S	[27,29]	“normal:5”, “warezclient:274”
9	N	[30,77]	“normal:285”

Table 6.1: Feature ranges of the discrete feature “hot”.

6.2 Building the NSA Signature Base

From the same training audit trails, we evaluate every instance as a signature. First, we replace the values in the instance with their corresponding feature ranges. The status of the instance is also accumulated within the signature. The NSA label of a signature is determined by its accumulated statuses. The signature is then added into the NSA signature base. Hence, the signature base consists of the normal signatures for SID, the attack signatures for SID, and the suspicious signatures.

If there exist ‘suspicious’ signatures, it implies that the features in the feature vector are not enough to distinguish the attack and normal behaviors in the audit trails. For this reason, the ‘suspicious’ signatures will lead to detection errors (i.e., false alarms or false negatives) in the detection results. In an intrusion detection system, the ‘suspicious’ signatures must be classified as ‘normal’ or ‘attack’ as they are not useful for the subsequent response strategies.

6.3 A Detection Mechanism

In the detection mechanism, the feature ranges of every feature and the NSA signature base are used to detect every instance in the test (or realtime) audit trails. To this end, every instance of feature vector is first extracted from the test audit trails, and every

CHAPTER 6. UNIFYING SIGNATURE-BASED AND ANOMALY-BASED INTRUSION DETECTION

of its values is replaced by its corresponding feature range to construct a temporary signature for the instance. We then use a simple logic to represent the detection results. If the temporary signature matches an identical signature in the NSA signature base, the detection result is given by the status list of the matched signature. Otherwise, the detection result is an ‘anomaly’. The outputted anomalies are further classified in Table 6.2 according to their root causes. We also include two special classes in the categorization for the scenario when the identical signature has the NSA label ‘A’ or ‘S’.

Anomaly Class	Meaning / Root Causes
D-I	the identified signature has the NSA label ‘A’. It is further classified as D-I-C or D-I-I depending on whether the test signature is identified correctly. For example, if the instance caused by ‘portsweep’ is identified as ‘portsweep’, it is D-I-C. Any other incorrect identification, such as ‘ipsweep’, ‘satan+portsweep’, will make it D-I-I.
D-S	the identified signature has the NSA label ‘S’.
D-A-I	the value of one feature does not fall into any of the predefined feature ranges. It is caused by the fact that we cannot get the exact feature space of a feature from the training audit trails.
D-A-II	the NSA labels of feature ranges in the temporary signature are contradictory, say, one is ‘N’ and another is ‘A’.
D-A-III	Excluding the above anomaly types, there is no identical signature in the NSA signature base.

Table 6.2: The anomaly classes in USAID. In a class name, ‘D’ represents that it is in the detection results, ‘-I’ represents that it is identified as an intrusion, ‘-S’ represents that it is identified by a suspicious signature, ‘-A’ represents that it is an anomaly.

There are many different root causes for the anomalies in the audit trails. Our anomaly categorization is based on the root causes of anomalies, so we believe that it could help to understand the detection results better. For example, one reason for $\mathcal{P}_A^3$ is that it is difficult to classify their anomalies in AID (cf., Chapter 4). Among these five anomaly classes, only **D-I-C** can be identified with correct intrusion name(s) but it does not exist in AID techniques.

6.4 Experimental Results on USAID

The dataset from KDD CUP 1999 is utilized in our experiments (cf., Chapter 3). There are two main portions in the experimental results. First, we show the characteristics of the NSA signature base. Secondly, the performance comparison between USAID and other existing techniques is performed.

6.4.1 Feature Ranges of Every Feature

In our USAID implementation, the feature ranges of every feature are collected from the training dataset along with their NSA labels. In our experiments, the average numbers of normal, suspicious and attack feature ranges of all features in the feature vector are $\overline{N} : \overline{S} : \overline{A} = 23 : 18 : 12$. Given the relatively large number of feature ranges in the ‘suspicious’ subspace, it is clear that only one feature from our selected feature vector cannot correctly distinguish between normal and attack behaviors. Therefore, the signature base in USAID, which assists the detection ability of every feature, is necessary to enhance its efficiency. Table 6.1 gives the feature ranges of the feature ‘hot’, in which the feature ranges are indexed, and the frequency information is also collected. To save space, each feature range is presented by its index in a signature.

6.4.2 The NSA Signature Base

With respect to the collected feature ranges, all signatures are built from the same training dataset. In any signature, each of its feature ranges is represented by its index and NSA label with a square bracket ‘[]’ (e.g. ‘[0,S]’), and the compounding between feature ranges is represented by ‘+’. A signature could be: ‘[0,S]+[1,S]+[24,S]+[9,S] =>N:{normal:980}’, in which the part ‘N:{normal:980}’ after ‘=>’ means that the NSA label of the signature is ‘N’, and that it occurs 980 times in the training audit trails as a *normal* instance. Though the NSA label of every feature range has no use in USAID, we keep it for future reference, such as in anomaly context identification [LD04b]. In addition, we have to determine whether an instance matching a suspicious signature is from an attack, so the frequency information of every signature is also stored with it. Besides that, there are two possible usages of the frequency information. First, it is useful to determine the parameter α in Chapter 4. Another application is to measure the detection confidence. Both applications are left for our future research.

6.4.2.1 Statistics of the Signature Base

In the NSA signature base, the numbers of the normal, suspicious and attack signatures are $N : S : A = 60371 : 58 : 2779$. Even though some suspicious signatures still exist, the detection ability has been improved much in comparison to any single feature in the feature vector (cf. Section 6.4.1). The existence of suspicious signatures indicates

that the selected features in our experiments (i.e., in the experimental datasets) are not enough to detect all known intrusions completely. Also, the high ratio of N and A , namely $N : A = 60371 : 2779 \approx 21.7$, is quite significant since it indicates that the detection speed of SID techniques will be faster for known intrusions.

In our experiments, the total of all possible signatures due to the cartesian products of the feature ranges of all 41 features are 8.38×10^{33} . It indicates that searching unknown intrusion signatures thoroughly via negative selection algorithm [HF00] from 8.38×10^{33} signatures is a mission impossible.

6.4.2.2 Signature Variations of a Behavior

We observed that most intrusions cause more than one signature. For example, for **portsweep**, the number of signatures is 941; **ipsweep**, 72; **satan**, 389. The observation shows that the variations of an intrusion do exist with a significant amount. Therefore, in SID, the difficulty of manually generating all signatures of intrusion variations leads to its intrusion variation detection problem.

We also found that some signatures are shared by several intrusions, such as **portsweep** and **neptune**. Even though the number of shared signatures is small (Table 6.3), this phenomenon shows that, under some scenarios (e.g., *inadequate no. of features*), it is difficult to identify these intrusions uniquely. Therefore, their response strategies must be designed carefully. If the response strategies for the intrusions with overlapped signatures are much different, then generating a response to such intrusions may lead to disastrous results.

6.4.2.3 Shared Signatures among Behaviors

Also in Table 6.3, we enumerated the numbers of signatures shared between different intrusion categories. The normal category will share many signatures with other intrusion categories. This is the main source of false alarms or false negatives in the detection phase. From this table, we can see that the signatures of the ‘R2L’ intrusions are not shared with any other intrusion category, whereas only the intrusions in the ‘probe’ category [Ken99] have shared signatures with every other category.

One possible reason for this phenomenon lies in the proportions of signatures in each category, that is, $normal : probe : DOS : U2R : R2L = 60432 : 1661 : 1255 : 65 : 42$. Another reason is that the major principles of the intrusions in the ‘probe’

CHAPTER 6. UNIFYING SIGNATURE-BASED AND ANOMALY-BASED INTRUSION DETECTION

category are similar to each other [Ken99], so the shared signatures within the ‘probe’ category are significant in Table 6.3. The strategies behind ‘probe’ and ‘DOS’ are similar, but it is different from the one behind ‘U2R’ and ‘R2L’. Therefore, ‘probe’ and ‘DOS’ can be classified in one class, and ‘U2R’ and ‘R2L’ in another class [Ken99]. The differences between these two classes explain why there are few signatures shared by them.

	normal	probe	DOS	U2R	R2L
normal	-	47	16	8	0
probe	47	33	10	2	0
DOS	16	10	0	0	0
U2R	8	2	0	0	0
R2L	0	0	0	0	0

Table 6.3: The statistics of shared signatures.

6.4.3 Detection Results

In the detection phase, we will evaluate the effectiveness of USAID in classifying an instance into ‘normal’ or one of the five anomaly categories. Figure 6.2 illustrates

	Normal	D-I-C	D-I-I	D-S	D-A-I	D-A-II	D-A-III
Normal	94.239	0	0.786	0.094	0.005	0.274	4.603
Known	0.222	93.527	0.054	0.0026	0.0004	0.091	6.103
New	1.82	0	12.8	0.08	0	3.48	81.8

Figure 6.2: Detection results of USAID (%).

the detection results for the **known** intrusions (which are in both training and test datasets) and for the **new** intrusions (which are only in the test dataset) as indicated in Table 3.4 (cf. Chapter 3), as well as the detection results for the **normal** instances. For an intuitive comparison, the detection performance is represented as percentage. Every row in this table includes the percentages of (*normal, known intrusion, or new intrusion*) instances detected as normal, D-I, D-S, D-A-I, D-A-II, D-A-III.

USAID can detect most, but not all, intrusions. Therefore, USAID should be deployed as one layer of a multi-layer integral security solution. The multiple layers need to complement one other in a sense that an undetected intrusion in one layer should be detected in another layer. In our experiments, USAID cannot detect two new intrusions, namely, **snmpgetattack** and **mailbomb**. Furthermore, the number of

instances of these two intrusions is so large that, if included, the failure to detect these two intrusions will virtually obscure the detection performance for other intrusions. With an expectation that mechanisms in other security layer can detect/prevent these two intrusions, we have eliminated them to evaluate whether USAID can detect other intrusions effectively. Essentially, the information in the chosen feature vector is not enough to distinguish these two new intrusions from normal behaviors. Thus, a possible solution is to add new features into the feature vector for USAID to detect these two intrusions.

6.4.3.1 Detection Performance

From Figure 6.2, it is clear that most known intrusions can be detected as *D-I-C*, and most new intrusions are detected as *D-A-III* in USAID. In addition, most normal instances have been correctly identified as normal with a small number of false alarms. Quantitatively, the false alarm rate is 1.451%, the detection rate for known intrusions 99.778%, the detection rate for most new intrusions (except `snmpgetattack` and `mailbomb`) 98.18%. In the detection results, some new intrusions have been detected as D-I-I. This is because some variations of new intrusions have the same signature as the known intrusions. This fact implies that the real intrusion may have intrusive characteristics, which are different from the intrusion(s) in the matched signature. Fortunately, the amount of the overlapping signatures is not that large.

6.4.3.2 Analysis on Detection Results

In Table 6.4 and Figure 6.2, there are only a small number of the instances in the test dataset that are detected as D-I-I, D-S, D-A-I, and D-A-II. From the statistics for D-I-I, it is clear that only a few known intrusions are misclassified as other intrusions, and that existing known intrusion signatures can be used to detect some new intrusions. The relatively high value (12.8%) of new intrusions detected as ‘D-I-I’ indicates the similarity between known and new intrusions with respect to the feature vector.

For D-S, its existence indicates that the selected feature vector and/or the training dataset are not enough to distinguish all instances, but fortunately, its number is not that large. For D-A-I, its negligible amount shows that the feature ranges of every feature generalized from the training dataset cover most of its feature spaces. The statistics for D-A-II indicates that the labeled NSA properties of feature ranges are

correct for most instances, and the relatively higher value for new intrusions in D-A-II (3.48%) proves the existence of $AS(\mu_i)$ and $NS(\mu_i)$ (c.f. Chapter 4 Section 4.2 for their meanings) in our formal framework.

6.4.3.3 Functions of Anomaly Categorization

	D-I	D-S	D-A-I	D-A-II	D-A-III
normal	476	57	3	166	2789
known intrusions	216853	6	1	212	14143
new intrusions	770	5	0	209	5306
false alarm rate	0.2%	83.8%	75.0%	28.3%	12.5%

Table 6.4: False alarm rates w.r.t. anomaly categories.

The different false alarm rates under different anomaly categories (in Table 6.4) show the effectiveness of the anomaly categorization in USAID. It is clear that the instance and/or the matched signature can be kept as an anomaly context for further investigation [LD04b]. Any signature detected as D-A-I, II, or III should be investigated via the anomaly context, and inserted into the NSA signature base with the correct status since it is a novel signature to USAID. With more signatures (normal and anomalous) investigated and inserted into the signature base, the false alarm rate will be lowered. The fact that the instances of an intrusion can match different signatures has shown again that intrusion variations are possible.

6.4.4 Performance Comparisons

As our experimental datasets is chosen from KDD'99 Classifier Learning Contest [Elk99], it is reasonable to evaluate the performance of USAID in comparison with participants of this contest. In [Elk99], every entry is assigned a cost at a time, and ultimately an average cost per test example is calculated for performance evaluation. The lower the average cost per test example, the higher rank the classifier. We use the same cost scheme in the performance comparisons.

The detection results of USAID in the test dataset are summarized in the following Table 6.5, in which the five behavior categories are numbered following [Elk99].

We only gave a slight comparison with participants in the KDD'99 contest because we have neglected two intrusions in our experiments. Its purpose is to show that, besides its aforementioned advantages, USAID is an effective solution for intrusion

	normal	probe	DOS	U2R	R2L	D-S, D-A-I/II/III
normal	57101	423	53	0	0	3015
probe	2	2341	7	0	4	1818
DOS	5143	14	215142	0	0	11167
U2R	8	0	0	0	0	69
R2L	7817	2	0	0	0	6939

Table 6.5: The detection results of USAID. The horizontal dimension is the predicted class of every test example, and the vertical dimension is its actual class. The first 5 columns constitute the confusion matrix of USAID, and the last column includes the total number of anomalies for all anomaly classes except D-I.

detection. In this comparison, the anomalies detected by USAID will be processed in two ways. First, these anomalies are categorized correctly to their actual categories. For example, the number of correctly predicated test examples of the category 1 is $2341+1818=4159$. In such case, the performance of USAID is scored 0.1355, which is much better than the 1st rank entry of KDD'99, 0.2331. Secondly, under the worst scenarios, these anomalies are categorized incorrectly into the categories with highest cost. For instance, if $\text{cost}(\text{R2L}, \text{normal})=3$ is the highest cost in the row 'R2L', the anomalies detected from actual category 'R2L' will be predicated as category 'normal', thus there are $7817+6939=14756$ test instances. The performance of USAID in the worst scenario is scored 0.3283, which is ranked 19th among all the participants. In summary, USAID is expected to achieve better performance in comparison with the participants of KDD'99 if the detected anomalies are categorized correctly.

6.5 Summary

In this chapter, the insights gained from our analysis in Chapter 4 and 5 are utilized to design a novel intrusion detection technique, called USAID (in Chapter 3). It incorporates all of available audit trails into its behavior models for intrusion detection. Our experimental results have shown that its detection performance is quite encouraging. Most new and known intrusions are detected correctly, and the false alarm rate is 1.451%. Comparison with other classifiers on the same dataset have indicated the superiority of USAID. Furthermore, our experimental results also shown that our anomaly categorization is useful since the false alarm rates under the various anomaly categories are different from each other.

CHAPTER 6. UNIFYING SIGNATURE-BASED AND ANOMALY-BASED INTRUSION DETECTION

We admitted that USAID will be threatened by mimicry attacks if all these feature ranges and NSA signature base are studied by an attacker. Nonetheless, getting all these information from a specific implementation of USAID is virtually impossible.

Chapter 7

varUSAID: USAID with Variable Length Signatures

As shown in Chapter 6, USAID gave promising detection performance in detecting known and unknown intrusions. However, USAID still suffers from the well known problem of *intensive computation* (i.e., $\mathcal{P}_{\mathbf{A}}^6$, see Chapter 4) in detecting behaviors. The computational cost becomes even more significant as the size of audit trails for intrusion detection keeps on growing with increasing internetworking applications.

In varUSAID, our aim is to improve the detection performance of USAID by reducing the computational cost in the detection process. We have applied two methods to achieve the objective. We first use feature selection to filter out insignificant features. Unlike the feature selection in [MS03, LS00], which use the overall detection performance in the *test* audit trails as the selection criteria, our feature evaluation is done by maximizing the distinguishing capability between normal and anomalous signatures only using the *training* audit trails (Section 7.2.2). To this end, we have introduced the concept of *variable-length signature* to further decrease the detection computation (Sections 7.1 and 7.2.3).

7.1 Variable-length Signatures

Within a signature, a feature range of its component feature may have the NSA label ‘N’ or ‘A’ by itself. This special characteristic can be utilized to reduce the computational cost in the detection phase. In other words, instead of using all the features in the feature vector, a new instance is detected/identified by such a component feature range with the NSA label ‘N’ or ‘A’.

Definition 7.8 (Variable-length Signature) Given $FV = \{F_1, F_2, \dots, F_n\}$ with labelled feature ranges for every feature, a *variable-length signature* is a feature range of $F_{1..i}$, such that its NSA label is either ‘N’ or ‘A’. The signature is called *variable length* as the number of features in it can vary from 1 to n , i.e., $1 \leq i \leq n$.

Example 7.3 Suppose that there is a feature vector $FV = \{ipaddr, port, service\}$, and the feature ranges of every feature are: $\Omega(ipaddr) = \{10.10.1.2, 10.10.2.2\}$, $\Omega(port) = \{22, 23, 80, 443\}$, $\Omega(service) = \{telnet, http, ftp, ssh\}$. A signature of the feature vector is $(10.10.2.2, 23, telnet)$. If the NSA label of the feature range $(10.10.2.2, 23)$ is ‘N’ or ‘A’, $(10.10.2.2, 23)$ is a variable-length signature.

The relation between signatures and variable-length signatures is subtle but important. A signature with the NSA label ‘N’ or ‘A’ can be reduced in size by omitting some component feature ranges if that does not affect its NSA label. Thus, it can become a variable-length signature. If a signature bears the NSA label ‘S’, it cannot shed any feature ranges, so that it remains of the fixed maximum length.

7.2 USAID with Variable-length Signatures

Figure 7.1: A general architecture for varUSAID.

The architecture of varUSAID consists of four modules (Figure 7.1). The first module extracts the three feature subspaces for every feature. The second module does the feature ranking and selection. The signature building module constructs the variable-length signature base, named as NSA signature base as in USAID. The last module incorporates a detection mechanism using variable-length signatures.

7.2.1 Feature Subspaces Extraction

The mechanisms in this module are the same to the first module in USAID. Please refer to Section 6.1 for details.

7.2.2 A Sorted Feature Vector

In this section, a function is first defined to measure the distinguishing ability of a feature, and then it is used to sort all features in the feature vector. To achieve it, the attack feature subspace is further split into two feature subspaces.

Two Further Feature Subspaces. Consider a compound feature $F_{1\dots i}$, with $N(F_{1\dots i})$, $S(F_{1\dots i})$ and $A(F_{1\dots i})$, and an atomic feature F_{i+1} with $N(F_{i+1})$, $S(F_{i+1})$ and $A(F_{i+1})$. Basically, it is possible that a feature range labelled ‘A’ is caused by several intrusions. The feature range that is labelled ‘A’ because of a single intrusion is extremely valuable for identifying that intrusion, and is thus highly desirable. For this reason, the attack feature subspace $A(F_{1\dots i})$ is split into two feature subspaces $A_n(F_{1\dots i})$ and $A_1(F_{1\dots i})$. If the status list of the feature range in $A(F_{1\dots i})$ only has one intrusion, it is classified as $A_1(F_{1\dots i})$, otherwise, as $A_n(F_{1\dots i})$. As a result, there are now four feature subspaces for $F_{1\dots i}$, namely $N(F_{1\dots i})$, $S(F_{1\dots i})$, $A_n(F_{1\dots i})$, and $A_1(F_{1\dots i})$, and for $F_{1\dots i+1}$: $N(F_{1\dots i+1})$, $S(F_{1\dots i+1})$, $A_n(F_{1\dots i+1})$, and $A_1(F_{1\dots i+1})$ respectively.

7.2.2.1 Measuring and Sorting Features

Figure 7.2: Four scenarios of the compounding between $F_{1\dots i}$ and F_{i+1} . The increase of the identification capacity is due to the compounding operations represented by red bold lines between F_{i+1} and $F_{1\dots i+1}$.

CHAPTER 7. varUSAID: USAID WITH VARIABLE LENGTH SIGNATURES

In Figure 7.2, the four compounding scenarios are illustrated to design a measure for ranking features. As indicated in (A), the compounding between feature ranges in $N(F_{1...i})$ and feature ranges in $\Omega(F_{i+1})$ will result in feature ranges in $N(F_{1...i+1})$. Thus, there is no improvement in the distinguishing ability for intrusion detection when we add more features to a compound feature range that is already labelled as ‘N’. Similarly, the compounding in (B) will not lead to any benefit. For this reason, we will not consider the compounding scenarios (A) and (B) in our feature ranking. In contrast, in the compounding scenarios (C) and (D), the feature ranges in $S(F_{1...i})$ and $A_n(F_{1...i})$ will increase the distinguishing ability of the new feature $F_{1...i+1}$ as some of their resultant feature ranges fall into $N(F_{1...i+1})$ or $A_1(F_{1...i+1})$. More feature ranges in $N(F_{1...i+1})$ and $A_1(F_{1...i+1})$ lead to better performance of varUSAID in detecting intrusions. In summary, to increase the distinguishing ability of the new feature, the increment due to compounding scenarios (C) and (D) should be maximized.

In addition, as our objective is to reduce the computation overhead in the detection phase, it is useful to consider frequency information of feature ranges in the training audit trails. The more frequently a variable-length signature occurs in the training audit trails, the smaller its length should be. To justify it, we assume that the training audit trails have the same frequency distribution as the test audit trails, and thus the detection computation can be reduced.

Therefore, the numbers of instances falling into $S(F_{1...i})$ and $A_n(F_{1...i})$ before the compounding with F_{i+1} but falling into $N(F_{1...i+1})$ and $A_1(F_{1...i+1})$ after the compounding, which represents the distinguishing ability of F_{i+1} , should be maximized. For the convenience of expression, the following notations are defined. Given the compounding between $F_{1...i}$ and F_{i+1} ,

- $\#S2N(F_{1...i}, F_{i+1})$ is the number of instances in $S(F_{1...i})$ that moves to $N(F_{1...i+1})$.
- $\#S21(F_{1...i}, F_{i+1})$ is the number of instances in $S(F_{1...i})$ that moves to $A_1(F_{1...i+1})$.
- $\#n21(F_{1...i}, F_{i+1})$ is the number of instances in $A_n(F_{1...i})$ that moves to $A_1(F_{1...i+1})$.

Using them, the relative distinguishing capability of F_{i+1} is $\Theta(F_{i+1}|F_{1...i})$:

$$\Theta(F_{i+1}|F_{1...i}) = \#S2N(F_{1...i}, F_{i+1}) + \#S21(F_{1...i}, F_{i+1}) + \#n21(F_{1...i}, F_{i+1}) \quad (\text{Eq. 7.1})$$

$\Theta(F_{i+1}|F_{1...i})$ represents the increase of distinguishing capability due to the feature F_{i+1} . The feature ranking is shown in Algorithm 4 using a greedy strategy. In step

Algorithm 4 Feature ranking.**Require:** Feature Vector FV , Number of Features n , Training audit trails Σ_{tr} ;

```

1:  $FV_{sort} = \emptyset$ ;
2: select a feature  $F_{max}$  with the max. instances falling into  $N(F_k)$  and  $A_1(F_k)$  ( $1 \leq k \leq n$ );
3: insert  $F_{max}$  into  $FV_{sort}$ ;
4: for  $i = 2$  to  $n$  do
5:    $\Theta_{max} = -\infty$ ;  $F_{\Theta_{max}} = -1$ ;
6:   for  $j = 1$  to  $n$  do
7:     if  $F_j \notin FV_{sort}$  then
8:       build the compound feature by compounding  $F_j$  and the features in  $FV_{sort}$ ;
9:       calculate  $\Theta(F_j|FV_{sort})$  from  $\Sigma_{tr}$ ;
10:      if  $\Theta(F_j|FV_{sort}) > \Theta_{max}$  then  $\Theta_{max} = \Theta(F_j|FV_{sort})$ ;  $F_{\Theta_{max}} = j$ ; endif
11:    endif
12:  end for
13:  insert  $F_{\Theta_{max}}$  into  $FV_{sort}$ ;
14: end for
15: output  $FV_{sort}$ ;
```

i , the feature that maximizes the value of $\Theta(F_i|FV_{sort})$ will be selected, and then the feature is appended into a so-called sorted feature vector FV_{sort} . In next $i + 1$ -th step, all features outside FV_{sort} are evaluated based on their values on $\Theta(F_{i+1}|FV_{sort})$. The ultimate sorted feature vector FV_{sort} , after evaluating all features in FV , will be used in the following two modules. The complexity of our algorithm is $O(n^2)$.

7.2.3 Building the NSA Signature Base

Definition 7.9 (Signature Base) *Assuming that there exist labeled training audit trails, and its feature vector is FV , a signature base Ω consists of signatures and variable-length signatures built from the training audit trails, so that every instance in the training audit trails will match one of its signatures or variable-length signatures. Specifically, the signature base consists of normal and attack variable-length signatures, and suspicious signatures built from the training audit trails.*

The NSA signature base building process starts with the first feature in FV_{sort} , and includes one additional feature in each step in the sorted order. At i -th step, only the compound feature ranges in $N(F_{1...i})$ and $A_1(F_{1...i})$ are left as variable-length signatures in the NSA signature base, and the compound feature ranges in $S(F_{1...i})$ and $A_n(F_{1...i})$ are carried forward for the subsequent compounding operation to increase the identification ability of the signature base. Finally, after compounding with the last feature, the compounding feature ranges in $S(FV_{sort})$ and $A_n(FV_{sort})$ are left as signatures in the signature base. Figure 7.3 illustrates the process of constructing

CHAPTER 7. varUSAID: USAID WITH VARIABLE LENGTH SIGNATURES

Figure 7.3: Building the NSA signature base in varUSAID.

Algorithm 5 Building the NSA signature base in varUSAID.**Require:** Sorted Feature Vector FV_{sort} , Number of Features m in FV_{sort} , Training audit trails;

- 1: NSABase= $\emptyset$;
- 2: **for** $i = 1$ to m **do**
- 3: build the compound feature $F_{1\dots i}$ in FV_{sort} ;
- 4: **for** each feature range in $N(F_{1\dots i})$ and $A_1(F_{1\dots i})$ **do**
- 5: **if** it does not match the signatures in NSABase **then**
- 6: insert it into NSABase as a variable-length signature;
- 7: **end if**
- 8: **end for**
- 9: **end for**
- 10: insert $S(F_{1\dots m})$ and $A_n(F_{1\dots m})$ to NSABase;
- 11: output NSABase;

the NSA signature base, and Algorithm 5 describes it in detail. Like the Huffman encoding, the NSA signature base thus constituted is compact enough to represent all the behavior identification information in the training audit trails.

VARIABLE-LENGTH SIGNATURES	NSA Labels	INTRUSIONS OR NORMAL
[94,A]	A	portsweep
[1069,N]	N	normal
[2,S]+[214,S]	A	buffer_overflow
[0,S]+[182,S]+[84,N]	N	normal
[0,S]+[86,S]+[0,S]	A	smurf
[0,S]+[277,S]+[239,S]+[14,S]	A	back

Table 7.1: Several examples of variable-length signatures.

Table 7.1 shows several example variable-length signatures. In every signature, the feature range is denoted by its index, and its NSA label is paired with the index using a pair of square brackets. The compounding of the feature ranges is denoted by ‘+’. For instance, in ‘[2,S]+[214,S]’, ‘[2,S]’ refers to the second range of the first feature in FV_{sort} (which is ‘source bytes’, please see the full description of FV_{sort} in the appendix), and ‘S’ indicates that its label is ‘suspicious’. Similarly, ‘[214,S]’ represent the 214th

CHAPTER 7. varUSAID: USAID WITH VARIABLE LENGTH SIGNATURES

range of the second feature (*‘service’*) in FV_{sort} , and the whole expression, along with its NSA label and status list, constitutes a complete variable-length signature “[2,S]+[214,S]=>A:buffer_overflow”. Even though the NSA labels of feature ranges are not useful in varUSAID, they are included in variable-length signatures for future usage, such as the anomaly context identification scheme [LD04b].

There usually exist ‘suspicious’ signatures (i.e., $S(F_{1...m}) \neq \emptyset$ in Algorithm 5). It implies that the feature vector is not enough to completely distinguish between intrusive and normal behaviors in the audit trails. Thus, these ‘suspicious’ signatures will lead to detection errors (*false alarms or false negatives*) in varUSAID.

Definition 7.10 (Signature Variations) *In the signature base, every element is a signature or a variable-length signature. If two of its elements A and B have the same status list (thus, the same NSA label), A is called a signature variation of B , and vice versa. For example, “[1069,N]=>N:Normal” and “[0,S]+[182,S]+[84,N]=>N:Normal” in Table 7.1 are signature variations since both are caused by ‘normal’ behavior(s).*

In the signature base, one behavior (*normal or an intrusion*) may lead to several (*variable-length*) signatures. This causes the well-known intrusion variation detection problem of SID. In general, the number of signature variations in the signature base indicates the diversity of the corresponding intrusions.

7.2.4 A Detection Mechanism via Variable-length Signatures

The detection mechanism using variable-length signatures is described as follows. An instance of a feature vector is first extracted from the test audit trails. In varUSAID, the variable-length signature $VarSig$ is generated incrementally from the test instance by mapping of the values one at a time. We then compare the signature thus generated with the variable-length signature base. If a match is found, the detection result is given based on the matched signature and the remaining features are not used (in lines 7-13). In building $VarSig$, there are two exceptions, in which ‘anomaly’ is detected: a value could fall outside the known feature subspace (in the 4th line), and the feature ranges in $VarSig$ may have contradict NSA labels (in the 6th line). This instance detection procedure reduces the computation overhead of the detection phase significantly.

Algorithm 6 Detecting an instance in varUSAID.**Require:** NSABase, FV_{sort} and its size m , An instance in the test audit trails;

```

1: DResult=nil; {detection result for the instance}
2: VarSig=nil; {variable-length signature corresponding to the instance}
3: for  $i = 1$  to  $m$  do
4:   if  $v_{F_i}$  falls outside the known feature space then DResult="Anomaly"; break; end if
5:   attach the corresponding feature range of  $F_i$  into VarSig;
6:   if the feature ranges in VarSig are contradictory then DResult="Anomaly"; break; end if
7:   if VarSig $\in$ NSABase then
8:     if VarSig is identified by an attack signature then
9:       DResult="Anomaly"; break;
10:    else if VarSig is identified by a 'normal' signature then
11:      DResult = "Normal"; break;
12:    end if
13:  end if
14: end for
15: if varSig is identified by a 'suspicious' signature then DResult="Anomaly"; end if
16: if DResult is 'nil' then DResult="Anomaly"; end if
17: output DResult;

```

7.2.5 Mimicry Attacks on varUSAID

The threat to varUSAID from the mimicry attacks (introduced by Wagner et al.[WS02]) can be explained as follows. In USAID, a mimicry attack achieve its goal mimicking all the feature ranges of a 'normal' signature. In contrast, a mimicry attack to varUSAID can be designed more easily than USAID since the short 'normal' variable-length signatures will be the preferred by attackers. In this aspect, varUSAID becomes more vulnerable than USAID.

7.3 Experimental Results on varUSAID

The dataset from KDD CUP 1999 is utilized in our experiments (c.f., Chapter 3). Briefly, there are two parts in the experimental results. First, we focus on mining the sorted feature vector, building the signature base by variable length signatures. Then, performance comparison is performed before and after utilizing variable-length signatures. To save space, we would refer the reader to Chapter 6 for the details of the feature ranges since they are the same as in USAID.

7.3.1 Mining a Sorted Feature Vector

Figure 7.4 illustrates the percentage of instances in $N(F_{FV_{sort}})$ and $A_1(F_{FV_{sort}})$ with more features being added into the sorted feature vector. Obviously, the curve con-

CHAPTER 7. varUSAID: USAID WITH VARIABLE LENGTH SIGNATURES

Figure 7.4: Percentage of instances identified by the increasing FV_{sort} .

verges to a stable value as more features are included into the sorted feature vector, i.e., more instances in the training audit trails are identified by these features.

For any further feature F_k in our applied feature vector, $\Theta(F_k|F_{FV_{sort}}) = 0$. That is, there are no instances in the training audit trails that will be identified by appending a further feature into the sorted feature vector. For this reason, the stopping criteria for feature selection in varUSAID is that the value of $\Theta(F_k|F_{FV_{sort}})$ is zero. As the minimum value of $\Theta(F_k|F_{FV_{sort}})$ is zero as well, the stopping criteria does guarantee that there is no additional distinguishingability left in the remaining features. Using this criteria, (m=)17 features are selected into the sorted feature vector, i.e., (n-m=)24 features are discarded before continuing with further experiments (for details, cf. the sorted feature vector in the appendix).

7.3.2 Building the NSA Signature Base

ITEMS	$N(\dots)$	$S(\dots)$	$A_n(\dots)$	$A_1(\dots)$	TOTAL
Size of NSA base in USAID	60371	43	15	2779	63208
Size of NSA base in varUSAID	952	43	15	1072	2082
Average len. of var.-length signatures	5.79	17	17	5.53	5.97

Table 7.2: The constitutions of NSA signature base in USAID and varUSAID.

Table 7.2 lists the statistics of the variable-length signatures in the NSA signature base. The average length of variable-length signatures in varUSAID is 5.97. Considering that the length of signatures in USAID is 41, varUSAID achieves 85.44% saving in the average length of a variable-length signature. In addition, the ratio of the signature base sizes is given by $|USAID_{base}| : |varUSAID_{base}| = 63208 : 2082 = 30.36$, i.e., the storage space of NSA signature base in varUSAID is cut down by 99.52% in

CHAPTER 7. varUSAID: USAID WITH VARIABLE LENGTH SIGNATURES

comparison with USAID. Note that, the size of $S(\dots)$ and $A_n(\dots)$ remain unchanged in USAID and varUSAID. The fact is consistent to our varUSAID design purpose to reduce the computation rather than to remove the ambiguity in the audit trails.

NAME/LENGTH	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
normal	141	214	28	29	62	37	67	4	76	265	1	17	5	2	0	0	8
neptune	1	0	23	45	0	147	3	43	9	0	0	7	3	0	1	1	0
buffer-overflow	1	10	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
portsweep	2	0	84	48	7	66	2	37	0	2	0	0	0	0	0	1	6
warezclient	3	19	0	1	0	2	7	0	3	37	1	0	0	0	0	0	0
back	19	8	3	10	3	2	0	0	12	0	0	5	0	0	0	0	0
pod	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
teardrop	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rootkit	0	2	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0
smurf	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
phf	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
warezmaster	0	1	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
multihop	0	4	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
satan	0	8	19	102	1	43	1	12	1	13	2	0	0	0	0	0	0
imap	0	3	0	1	0	1	0	0	2	0	0	0	0	0	0	0	0
ftp-write	0	5	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0
ipsweep	0	1	2	1	6	12	20	0	2	8	0	0	0	0	0	0	0
nmap	0	0	52	0	0	1	3	0	1	1	0	0	1	0	0	0	0
guess-passwd	0	0	1	1	8	0	0	0	0	1	0	0	0	0	0	0	0
land	0	0	0	1	0	0	3	0	0	0	0	0	1	0	2	0	0
load module	0	0	0	0	0	2	1	0	0	6	0	0	0	0	0	0	0
spy	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
perl	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0

Table 7.3: Number of signature variations in the training audit trails.

Signature Variations. Table 7.3 summarizes the signature variations in the NSA signature base. It is clear that the same intrusion can lead to different variable-length signatures. Conversely speaking, one intrusion signature is not enough to detect all intrusive instances for most intrusions, which is a hard problem for SID (P_S^3). As the signature variations of an intrusion have different lengths, there is no specific feature vector to detect all intrusion variations efficiently, i.e., different feature vectors are needed to detect some intrusion variations. For example, to detect intrusion ‘back’, only 19 intrusion variations are detected if only the first feature ‘source bytes’ is utilized, and the number of detected intrusion variations becomes 30 if the first 3 features are used.

The ‘normal’ behavior is the most diverse behavior, which has the largest number of ‘normal’ signature variations. This phenomenon can be explained as follows. In the normal usage of a computing resource, there are many tasks to do with different objectives, and different tasks lead to different signatures. However, in an intrusion, the objective of the task is to attack and compromise a computing resource, so the

intrusive task is not as complicated as normal tasks. Therefore, an intrusive behavior causes fewer signatures than ‘normal’ behavior.

7.3.3 Detection Results

Our experimental purpose is to show the effect of the feature ranking method and variable-length signatures. We evaluated varUSAID in comparison with USAID that, comparing with KDD Cup participants [Elk99], has shown its superiority for intrusion detection in Chapter 6. In order to achieve it, three experiments are designed: *USAID with 41 features (USAID-41)*, *USAID with the selected 17 features (USAID-17)*, and *varUSAID with the sorted feature vector (varUSAID)*. Similar to USAID in Chapter 6, two intrusions, `snmpgetattack` and `mailbomb`, are eliminated from the detection results.

In Table 7.4, they are compared with respect to four parameters in the detection results: (1)**FAR**: false alarm rate in detecting all intrusions; (2)**DR-Known**: detection rate in detecting known intrusions; (3)**DR-New**: detection rate in detecting new intrusions; (4)**Num-FR**: number of feature ranges in detecting an instance.

PARAMETERS	FAR (%)	DR-Known (%)	DR-New (%)	Num-FR
USAID-41	1.451	99.779	98.184	41
USAID-17	1.409	99.778	97.901	17
varUSAID	0.431	95.928	55.172	3.205

Table 7.4: Performance parameters in our experiments.

USAID-41 and USAID-17 have almost the same detection rate and false alarm rate, which indicates the effectiveness of our feature ranking. The slight difference between USAID-41 and USAID-17 is caused by a similar reason as the one causing the difference between USAID and varUSAID, which is explained below.

Intrusion detection is generally known as a classification problem [LSM99, LS00]. In any classification problem, one needs to look for a suitable tradeoff between detection rate and false alarm rate, as improving one usually worsens the other. Between the lower detection rate of varUSAID and the higher false alarm rate of USAID, the former is the lesser of the two evils for the purpose of building an effective IDS, because of the well-known problem of base-rate fallacy [Axe99]. In addition, for real-time implementation, varUSAID is a much better option because of its significantly lower computational cost. These points are further elaborated below.

In both USAID and varUSAID, a test instance can be formalized as a signature by replacing its feature values with corresponding feature ranges. For every signature in USAID, there is one corresponding variable-length signature in varUSAID, in which some feature ranges are discarded for compactness. However, some new intrusions will cause anomalies in the discarded feature ranges. For example, assuming that one signature in USAID is “[0,N]+[14,S]+[23,S]=>N:Normal” and the corresponding variable-length signature in varUSAID is “[0,N]=>N:Normal”. If the signature of an instance is “[0,N]+[25,S]+[23,S]”, it will be detected as ‘normal’ in varUSAID but as ‘anomaly’ in USAID. For this reason, although the longer signature increases the detection computation, retaining the redundancy in a signature (i.e., the insignificant feature ranges) is useful to enhance the detection performance of novel intrusions.

However, for intrusion detection, a lower false alarm rate is more desirable than the sacrificed detection rate. The fact can be explained using the following two examples. Suppose that an intrusion signature can detect an intrusion without false alarms, but it is unable to detect other intrusions and variations, and consequently, has a low detection rate. This sort of signatures are usable (and already deployed) in a real system like Symantec Antivirus [Szo05]. In comparison, a signature is useless if by increasing its detection rate leads to more false alarms, which are intolerable to the end-user because of the base-rate fallacy [Axe99]. In other words, a security system will become useless just because the end-user is frustrated by too many false alarms. Furthermore, the slightly reduced detection rate can be compensated for in the deployment of varUSAID. Like USAID (in Section 6.4.3.1), varUSAID is also deployed as one layer in an integral security solution, where all layers complement each other for intrusion detection.

In summary, the lower false alarm rate and lower number of feature ranges in detection are very encouraging since it increases the usability of a system and lessens its computation overhead significantly.

7.4 Summary

In this chapter, a solution is proposed to cut down the expensive detection computation, in which we introduce a method for feature selection and a novel concept of *variable-length signature*. Among our experimental results, we found that our feature

CHAPTER 7. varUSAID: USAID WITH VARIABLE LENGTH SIGNATURES

selection method is effective in incorporating all distinguishing ability in the sorted feature vector. Our experimental results also have shown that the false alarm rate in varUSAID was only 30% of that in USAID, that the storage space of the behavior model in varUSAID was only 0.48% of that in USAID, and that the computation in detecting intrusions has been decreased 92.18% w.r.t. USAID-41, and 81.15% w.r.t. USAID-17. Besides these advantages, the detection performance of varUSAID for known intrusions are also encouraging.

Most importantly, an important fact is discovered for the first time that the redundancy in signatures, which is removed by varUSAID, is useful to detect new intrusions.

Chapter 8

Automatic Signature Assemblers

Most intrusion detection products in the market are based on SID techniques using intrusion signatures. For instance, open source Snort [Roe99], commercial RealSecure [Rea]. However, most intrusion signatures utilized by these intrusion detection systems are manually constructed [NKS05], and there lacks a systematic methodology to build these intrusion signatures automatically. In addition, the privacy of the audit trails is increasingly an issue, so it is necessary to construct privacy-preserved signatures automatically instead of building them manually.

In this chapter, we present a new mechanism to build privacy-preserved intrusion signatures automatically. Based on the formal signature definition in our framework, we propose a new concept of *automatic signature assemblers* to build the behavior signatures on the fly.

8.1 Automatic Signature Assemblers

Figure 8.1: The workflow of automatic signature assemblers.

In our framework (Chapter 3), the signatures can be built automatically provided that the feature ranges of all features are defined beforehand. Using this property, we

design a signature assembly line for intrusion detection as illustrated in Figure 8.1, in which automatic signature assemblers, who have *no knowledge* about intrusions and intrusion detection, construct the behavior signatures for intrusion detection.

The signature assembly line consists of three modules. The **first** module collects the audit trails for intrusion detection. For example, Libpcap[Lib], winpcap[Win] etc.. With respect to a predefined feature vector, the **second** module extracts instances from the collected audit trails. Automatic signature assemblers will then construct a signature from an instance by replacing the values in the instance with their corresponding feature ranges, which have been well-defined beforehand. In addition, the feature ranges are further replaced by their indices to provide privacy-protection. In the **last** module ‘*signature analysis*’, the signatures are postprocessed.

An important step in the assembly line is to develop effective post-processing techniques for signature analysis. Several existing works can act as such techniques. For example, the signature was labelled by an expert *after the event*, and then it was used to fine-tune the behavior models for intrusion detection on the fly [LD04b]. The signatures thus generated were correlated to build attack scenarios [NXHA04]. In addition, the statistical analysis on signatures was also applied to study the characteristics of the audit trails, and further to identify the anomalies [AC03], and so on.

It is obvious that the first and last modules could be performed automatically. To make the assembly line automatic, it is important to automate the second module. A feature vector should be determined beforehand. Most importantly, in order to eliminate uncertainty in the assembly line, the feature ranges of every feature F should be collected in advance such that $R_F^1 \cup R_F^2 \cup R_F^3 \cup \dots = \text{Dom}(F)$. In the following sections, we will study how to collect such feature ranges for assemblers.

8.1.1 Discovering Feature Ranges for Assemblers

For numerical feature values, only a subset of the possible values are obtained from the training audit trails. The anomalousness of a novel value (i.e., *its NSA label in our framework*) is inferred from existing values. However, for a nominal feature, it is difficult to do such inference since all values are independent. Therefore, each of known nominal feature values is considered as a feature range by itself, and *a special feature range R_F^0 is assigned if a new value occurs*. In other words, our feature range discovery method is mostly applied to discrete and continuous features.

Inferring the NSA label of the new value can be performed using NSA labels of the neighboring values. This, in turn, requires that we should have a clear and unambiguous way to determine whether two values are neighboring. We use the following definition to determine the neighborhood of two (numerical) values:

Definition 8.11 *Two numerical values v_1 and v_2 are **neighbors** if there are no value falling between them in the training audit trails.*

Next, we address the question how to assign a NSA label to the new value given its left and right neighbors v_1 and v_2 . The answer lies in building the feature ranges in a systematic manner so that the space (v_1, v_2) is completely covered by one or more suitably labeled feature ranges. We first collect all values, $v_1, v_2, \dots, v_l$, and their NSA labels. Subsequently, the feature ranges are determined and labeled using several different methods as discussed below.

8.1.1.1 Methods to Build Feature Ranges

Suppose there are two neighboring values v_i and v_{i+1} ($i < l$), the NSA label of the intervening space (v_i, v_{i+1}) could be determined by $L(v_i)$ and/or $L(v_{i+1})$. If $L(v_i) = L(v_{i+1})$, the space $[v_i, v_{i+1}]$ will be assigned the same NSA label as $L(v_i)$. Otherwise, the space $[v_i, v_{i+1}]$ will be split into two parts, LS_i and US_i , based on different splitting strategies (thereof, different inference assumptions as in Chapter 5): **S1**: *equal splitting*; **S2**: *frequency-based splitting*; **S3**: *intrusion-constant splitting*.

These splitting strategies are defined below. The NSA label of LS_i is assigned as $L(v_i)$, and the NSA label of US_i is $L(v_{i+1})$. The total size in US_i and LS_i is $v_{i+1} - v_i + \varepsilon$ if these two parts are not overlapping, where ε is the value precision of the feature.

Different from the generalization evaluation in Chapter 5, we evaluate the significance of overlapping feature ranges by loosing the 2nd condition in section 3.2.2.1. This is due to the common argument that overlapping feature ranges could detect more intrusion variations in practice. Our experimental results in section 8.2.1.1 shown that the argument is invalid. We use a parameter O_s that determines the amount of overlapping around the splitting boundary between two neighboring feature ranges. A zero value for O_s indicates non-overlapping boundary between neighboring ranges.

8.1.1.2 S0: No Splitting.

For comparison, we will evaluate an extreme scenario when every value is simply considered as a feature range. This scenario is regarded as a baseline for the usefulness of assigning and splitting unknown feature subspace since we have not conducted any generalization in building feature ranges.

8.1.1.3 S1: Equal Splitting.

If $L(v_i) \neq L(v_{i+1})$, the splitting point is $\frac{v_i+v_{i+1}+\varepsilon}{2}$. Splitting with this point, the two parts are $LS_i = [v_i, \frac{v_i+v_{i+1}+\varepsilon+O_s}{2})$ and $US_i = [\frac{v_i+v_{i+1}+\varepsilon-O_s}{2}, v_{i+1}]$.

8.1.1.4 S2: Frequency-based Splitting.

We also collect the frequency information of all values in the training audit trails, $Fr_{v_1}, Fr_{v_2}, \dots, Fr_{v_l}$. If $L(v_i) \neq L(v_{i+1})$, the splitting point is $v_i + \varepsilon + (v_{i+1} - v_i - \varepsilon) * \frac{Fr_{v_i}}{Fr_{v_i} + Fr_{v_{i+1}}}$. Splitting with this point, the two parts are $LS_i = [v_i, v_i + \varepsilon + (v_{i+1} - v_i - \varepsilon) * \frac{Fr_{v_i}}{Fr_{v_i} + Fr_{v_{i+1}}} + O_s) * \frac{Fr_{v_i}}{Fr_{v_i} + Fr_{v_{i+1}}})$ and $US_i = [v_{i+1} - (v_{i+1} - v_i - \varepsilon + O_s) * \frac{Fr_{v_{i+1}}}{Fr_{v_i} + Fr_{v_{i+1}}}, v_{i+1}]$.

8.1.1.5 S3: Intrusion-constant Splitting.

Assume that there is a constant I_c . If $L(v_i) \neq L(v_{i+1})$, it is split and the two parts LS_i and US_i are labeled as follows. (1) $LS_i = [v_i, v_{i+1} - I_c - O_s)$ and $US_i = [v_{i+1} - I_c, v_{i+1}]$ if $L(v_i) = 'N'$ and $L(v_{i+1}) = 'A'$; and (2) $LS_i = [v_i, v_i + I_c)$ and $US_i = [v_i + I_c - O_s, v_{i+1}]$ if $L(v_i) = 'A'$ and $L(v_{i+1}) = 'N'$. Under other situations, the unknown part will be split equally as in *S1: equal splitting*.

The overlapping parameter O_s may lead to an incorrect LS_i or US_i whose boundaries are outside the interval $[v_i, v_{i+1}]$. These boundaries are manipulated to make sure that the splitting is applied only to the interval $[v_i, v_{i+1}]$.

Figure 8.2 illustrates one example of splitting an unknown feature subspace using S1, S2 and S3 with $O_s = 0$, $\varepsilon = 1$. There are two feature values $v_1 = 10$ and $v_2 = 15$, and $Fr_{v_1} = 1$, $Fr_{v_2} = 3$, $L(v_1) = 'N'$ and $L(v_2) = 'A'$. Using S1 in (b), the splitting point is $\frac{v_i+v_{i+1}+1}{2} = 13$, $LS_1 = [10, 13)$ and $US_1 = [13, 15]$. Using S2 in (c), the splitting point is $v_i + \varepsilon + (v_{i+1} - v_i - \varepsilon) * \frac{Fr_{v_i}}{Fr_{v_i} + Fr_{v_{i+1}}} = 12$, $LS_1 = [10, 12)$ and $US_1 = [12, 15]$. Using S3 with $I_c = 1$ in (d), $LS_1 = [10, 14)$ and $US_1 = [14, 15]$.

Furthermore, two feature ranges are merged if they are neighboring with the same NSA label. The merging is used to decrease the number of feature ranges. Specifically,

Figure 8.2: Splitting strategies for an unknown feature subspace.

the two neighboring ranges R_i and R_{i+1} , if $L(R_i) = L(R_{i+1})$, is merged into a feature range. That's also why the unknown subspace is split only if $L(v_i) \neq L(v_{i+1})$ in our splitting strategies. In Chapter 5, we have found that the merging step is useful in enhancing the detection performance, so it is used directly without further evaluation.

Evaluations. After building the feature ranges, we evaluate the different splitting strategies using a labeled test dataset. The test instance in the dataset feeds into the signature assembly line, and the outputted signature will be detected to determine whether it is in the behavior models. Based on the number of instances that are identified correctly, we can determine the usefulness of the splitting strategies.

8.2 Experiments: A Case Study

The dataset from KDD CUP 1999 is utilized in our experiments (cf., Chapter 3). Like the model generalization evaluation (in Chapter 5), we randomly sample 10000 instances from the total 4898431 training instances and 500 instances from the total 311029 test instances. We gave three pairs of such training and test samples, as well as the complete datasets, to make our experiments convincing.

8.2.1 Evaluating S0~S3 Splitting Strategies

Table 8.1 gives the notations used in this section to describe the detection performance.

Table 8.2 shows the detection performance when there is no label assigning and splitting for unknown feature subspaces. Obviously, most instances in the three test samples as well as the whole test dataset cannot be identified as their original labels, especially for normal behaviors. Specifically, only few normal instances are identified

CHAPTER 8. AUTOMATIC SIGNATURE ASSEMBLERS

INDEX	NOTATIONS	ORIGINAL CLASS	DETECTION RESULTS
1	$\#_{NN}$	normal	normal
2	$\#_{NA}$	normal	anomaly
3	$\#_{II}$	intrusion	original intrusion
4	$\#_{IA}$	intrusion	anomaly
5	$\#_{IN}$	intrusion	normal

Table 8.1: Notations used in detection results.

Sample Pair	Normal	Intrusion	$\#_{NN}$	$\#_{NA}$	$\#_{II}$	$\#_{IA}$	$\#_{IN}$
Pair 1	103	397	0	103	203	193	1
Pair 2	91	409	0	91	216	193	0
Pair 3	108	392	5	103	193	198	1
Complete	60953	250436	7669	52924	201537	45640	3259

Table 8.2: S0: without label assigning and splitting.

correctly, but more than half intrusive instances can be identified correctly, and there are fewer false negatives in comparison. Using the splitting strategy **S0**, an instance will be identified if it is identical to an existing instance in the training audit trails.

8.2.1.1 S1,S2,S3 Splitting Strategies.

Testing on Samples. In comparison with Table 8.2, the overall detection performance has been enhanced after applying **S1/S2/S3** splitting strategies in Table 8.3 and Table 8.4. For normal behaviors, the performance enhancement almost keeps the same for different splitting strategies. Although the performance enhancement for intrusive behaviors is a little different, it is too small to be negligible in comparison with the overall detection performance, especially for the sample pair 3.

There are two possible reasons for the small difference between **S1/S2/S3**. First, in the training audit trails, there are very few unknown feature subspaces for these three splitting strategies. Secondly, in the test audit trails, there are few instances whose feature values fall into the unknown feature subspaces. If these *two factors* were not present, we could conclude with certainty that the splitting strategies **S1/S2/S3** are indistinguishable in their effectiveness in discovering feature ranges. This conclusion will further benefit our framework since we have no need to incorporate any unstable distribution-related information into our framework. Further experiments in our case study show that the average number of unknown feature subspaces is 25 per feature in the complete training audit trails, and there are 398 instances that have values falling into the unknown feature subspaces in the complete test audit trails. These factors strengthen our conclusions.

CHAPTER 8. AUTOMATIC SIGNATURE ASSEMBLERS

(Pair 1) Normal:Intrusion=103:397

S?	I_c	O_s	$\#_{NN}$	$\#_{NA}$	$\#_{II}$	$\#_{IA}$	$\#_{IN}$
S1	-	0	35	68	280	115	2
S1	-	1	35	68	281	114	2
S1	-	2	35	68	281	114	2
S1	-	3	35	68	281	114	2
S1	-	5	35	68	281	114	2
S1	-	10	35	68	281	114	2
S2	-	0	35	68	280	115	2
S2	-	1	35	68	280	115	2
S2	-	2	35	68	280	115	2
S2	-	3	35	68	280	115	2
S2	-	5	35	68	280	115	2
S2	-	10	35	68	280	115	2
S3	0	0	35	68	280	115	2
S3	1	0	35	68	280	115	2
S3	2	0	35	68	281	114	2
S3	3	0	35	68	280	115	2
S3	5	0	35	68	280	115	2
S3	10	0	35	68	281	114	2
S3	0	1	35	68	280	115	2
S3	0	2	35	68	280	115	2
S3	0	3	35	68	280	115	2
S3	0	5	35	68	280	115	2
S3	0	10	35	68	280	115	2

(Pair 2) Normal:Intrusion=91:409

S?	I_c	O_s	$\#_{NN}$	$\#_{NA}$	$\#_{II}$	$\#_{IA}$	$\#_{IN}$
S1	-	0	39	52	294	113	2
S1	-	1	39	52	294	113	2
S1	-	2	39	52	294	113	2
S1	-	3	39	52	294	113	2
S1	-	5	39	52	293	114	2
S1	-	10	39	52	292	115	2
S2	-	0	39	52	292	115	2
S2	-	1	39	52	292	115	2
S2	-	2	39	52	292	115	2
S2	-	3	39	52	292	115	2
S2	-	5	39	52	292	115	2
S2	-	10	39	52	292	115	2
S3	0	0	39	52	290	117	2
S3	1	0	39	52	291	116	2
S3	2	0	39	52	291	116	2
S3	3	0	39	52	292	115	2
S3	5	0	39	52	294	113	2
S3	10	0	39	52	294	113	2
S3	0	1	39	52	290	117	2
S3	0	2	39	52	290	117	2
S3	0	3	39	52	290	117	2
S3	0	5	39	52	290	117	2
S3	0	10	39	52	290	117	2

(Pair 3) Normal:Intrusion=108:392

S?	I_c	O_s	$\#_{NN}$	$\#_{NA}$	$\#_{II}$	$\#_{IA}$	$\#_{IN}$
S1	-	0	45	63	273	115	4
S1	-	1	45	63	273	115	4
S1	-	2	45	63	273	115	4
S1	-	3	45	63	273	115	4
S1	-	5	45	63	273	115	4
S1	-	10	45	63	273	115	4
S2	-	0	45	63	273	115	4
S2	-	1	45	63	273	115	4
S2	-	2	45	63	273	115	4
S2	-	3	45	63	273	115	4
S2	-	5	45	63	273	115	4
S2	-	10	45	63	273	115	4
S3	0	0	45	63	273	115	4
S3	1	0	45	63	273	115	4
S3	2	0	45	63	273	115	4
S3	3	0	45	63	273	115	4
S3	5	0	45	63	273	115	4
S3	10	0	45	63	273	115	4
S3	0	1	45	63	273	115	4
S3	0	2	45	63	273	115	4
S3	0	3	45	63	273	115	4
S3	0	5	45	63	273	115	4
S3	0	10	45	63	273	115	4

Table 8.3: Evaluating S1, S2 and S3 splitting strategies.

CHAPTER 8. AUTOMATIC SIGNATURE ASSEMBLERS

(Complete) Normal:Intrusion=60593:250436

S?	I_c	O_s	$\#_{NN}$	$\#_{NA}$	$\#_{II}$	$\#_{IA}$	$\#_{IN}$
S1	-	0	57102	3491	215834	21636	12966
S1	-	1	57103	3490	215834	21636	12966
S1	-	2	57103	3490	215834	21636	12966
S1	-	3	57103	3490	215834	21636	12966
S1	-	5	57104	3489	215834	21636	12966
S1	-	10	57104	3489	215834	21636	12966
S2	-	0	57104	3489	215832	21638	12966
S2	-	1	57106	3487	215836	21634	12966
S2	-	2	57106	3487	215836	21634	12966
S2	-	3	57106	3487	215836	21634	12966
S2	-	5	57106	3487	215836	21634	12966
S2	-	10	57106	3487	215836	21634	12966
S3	0	0	57106	3487	215829	21641	12966
S3	1	0	57103	3490	215833	21637	12966
S3	2	0	57103	3490	215833	21637	12966
S3	3	0	57103	3490	215833	21637	12966
S3	5	0	57103	3490	215833	21637	12966
S3	10	0	57104	3489	215831	21639	12966
S3	0	1	57106	3487	215829	21641	12966
S3	0	2	57106	3487	215829	21641	12966
S3	0	3	57106	3487	215829	21641	12966
S3	0	5	57106	3487	215829	21641	12966
S3	0	10	57106	3487	215829	21641	12966

Table 8.4: Evaluating S1, S2 and S3 splitting strategies on the whole dataset.

Testing on the Complete Dataset. Comparing the performance enhancement between the test samples in Table 8.3 and the whole dataset in Table 8.4, we could make the same conclusion about the usefulness of these splitting strategies. So, the influence of incompleteness in the training audit trails on our conclusions is very small.

In summary, instead of regarding every occurring feature value as a feature range, we should split the unknown feature subspaces to enhance the usefulness of feature ranges for signature assemblers. Thus, we can identify more instances in the detection phase. However, in our case study, we also found that the overlapping scheme has no use in collecting feature ranges, and that the difference of these three splitting strategies is so insignificant that it can be negligible.

8.3 Summary

In this chapter, we proposed a signature assembly line to automatically generate behavior signatures for intrusion detection. It can build an intrusion signature in

CHAPTER 8. AUTOMATIC SIGNATURE ASSEMBLERS

comparison with the traditional signature building method. As another attractive characteristic, it is able to provide privacy protection for the audit trails.

We have evaluated how to collect the signature components for signature assemblers. The experimental results shown that the assignment of NSA label to unknown feature spaces is necessary for behavior identification. Although splitting of uncovered feature ranges improves performance, it is relatively insensitive to any particular splitting strategy as well as the allowance of overlapping feature ranges. The conclusions remain unchanged for varying degree of completeness of training audit trails. One future work is to propose post-processing techniques for the signature analysis.

Chapter 9

Anomaly Context Identification

As of now, most intrusion detection techniques are largely ineffective for use in an intrusion detection system (IDS) [Axe99]. This is due to the base-rate fallacy “*that in order to achieve substantial values of the Bayesian detection rate $P(\text{Intrusion}|\text{Alarm})$, we have to achieve a (perhaps in some cases unattainably) low false alarm rate*” [Axe99]. In other words, the higher false alarm rate will make a security site officer (SSO) frustrated, so that soon she would start to ignore all (*true and false*) alarms.

In AID techniques, it is difficult to determine whether an anomaly is caused by intrusions ($\mathcal{P}_A^3$). Thus, the quality of alarms triggered by anomalies cannot be guaranteed. The false alarm rate is further deteriorated because most behavior models of intrusion detection are incomplete. In this chapter, we introduce the concept of *anomaly context identification (ACI)*, which will help to filter the false alarms and to make up for the knowledge deficiency in the behavior models (i.e., model incompleteness). In ACI, the anomaly contexts of alarms are automatically extracted from the audit trails. After a careful study of the anomaly contexts, a SSO can distinguish the false positives¹ from the true positives within the detection results. Using this information, false alarms can be filtered out from all alarms to a significant extent. This information can then be used to improve/modify the intrusion detection engines as a feedback. As a result, the detection performance is enhanced ultimately.

Although our experiments performed are based on stide [HFS98] and uses system logs of the intruded system, the concept is more general and applicable to most AID methods (c.f. Section 9.4), and audit trails of network traffic.

¹False positives and false alarms will be used alternatively in this thesis.

9.1 Anomaly Context Identification

Let us first define the (minimal) anomaly context of an alarm.

Definition 9.12 (Anomaly Context) *In AID, the **anomaly context** of an alarm is the part of audit trails falling outside the normal behavior model that causes the alarm to be raised.*

Example 9.4 *Assume that the normal behavior model in stide [HFS98] with detector window $\omega = 3$ is $\{1-2-3, 3-1-2, 3-2-4\}$, and a test trace is ‘1-2-3-4’. In the test trace, ‘1-2-3’ is detected as ‘normal’ and thus there is no alarm. But the second 3-gram ‘2-3-4’ will be detected as an alarm, so the anomaly context of the alarm is ‘2-3-4’.*

In general, any larger context that contains an anomaly context of an alarm as a subcontext will also be an anomaly context of the same alarm. For instance, similar to ‘2-3-4’ in the above example, ‘1-2-3-4’ is also an anomaly context of the alarm.

Definition 9.13 (Minimal Anomaly Context) *An anomaly context of an alarm is minimal if any of its subcontext resulting from any further reduction of information from the context is not recognizable as anomalous, and consequently, no alarms are raised by them.*

Example 9.5 *Take the same scenario as in example 9.4. For the anomaly context ‘2-3-4’ of an alarm, suppose that the subcontext ‘2-3’ can also lead to an alarm with detector window $\omega = 2$ but ‘2’ or ‘3’ cannot lead to an alarm by stide detector with length $\omega = 1$. Then, ‘2-3’ is a minimal anomaly context of the alarm, but ‘2-3-4’ is not a minimal anomaly context.*

Since a minimal anomaly context includes all necessary knowledge about an alarm, identifying minimal anomaly context is useful to study the alarm. However, to identify the minimal anomaly context, the computational complexity of the ACI scheme will increase to a large extent. Even worse, most AID techniques lack formal foundations to define what constitutes the minimal anomaly context of an alarm. For above reasons, we only propose to identify the anomaly contexts in our ACI design, though the minimal anomaly context is mined in our exemplar ACI scheme.

9.1.1 Anomaly Contexts of an Intrusion

In general, every run of an intrusion can trigger more than one alarm, and there is a corresponding anomaly context for every alarm. Thus, more than one anomaly context can be identified corresponding to every run of the intrusion. To further evaluate the usefulness of these anomaly contexts, we classify them as *stable* or *transient* as follows.

Definition 9.14 (stable anomaly context) *An anomaly context is **stable** for an intrusion if the same alarm will be triggered in every run of the intrusion.*

Definition 9.15 (transient anomaly context) *An anomaly context is **transient** for an intrusion if the alarm is not triggered in a run of the intrusion.*

Example 9.6 *Suppose that training audit trails contain ‘1-2-3-2-1’, one run of an intrusion leave the audit trail ‘1-2-1-2-3-1’ and another run, ‘1-2-1-2-3-2’. If the detector window $\omega = 3$, the normal behavior model for stride is $\{1-2-3, 2-3-2, 3-2-1\}$. Thus, the anomaly contexts for the first run are $\{1-2-1, 2-1-2, 2-3-1\}$, and the anomaly context for the second run are $\{1-2-1, 2-1-2\}$. Following the above two definitions, the **stable** anomaly contexts are $\{1-2-1, 2-1-2\}$, and the **transient** one is $\{2-3-1\}$.*

The task of intrusion detection is to detect intrusions under varying environments. Therefore, the stable anomaly contexts are more useful for intrusion detection because they indicate an occurrence of an intrusion definitely among different runs of an intrusion under varying environments. A transient anomaly context is specific for every run of the intrusion, being determined by its specific environments, and thus it is less useful for intrusion detection.

The concept of anomaly context could be extended to an alarm in SID techniques. Since the intrusion signatures in SID are extracted from the audit trails of corresponding intrusions after the event, these intrusion signatures can be considered to be anomaly contexts of the alarms caused by the intrusions. Conversely, the anomaly contexts, which are identified from the audit trails caused by real intrusions, can be regarded as intrusion signatures in SID.

9.1.2 Effectiveness of ACI

Let us first discern what causes false alarms in AID and SID. As illustrated by Figure 4.1 in Chapter 4, the part of the intrusive behavior model that encroaches into the normal behavior space is the root cause for the false alarms in SID, and the part of the normal behavior space that is not covered by the normal behavior model is the root cause for the false alarms in AID.

In SID, the intrusive behavior model is built from the audit trails of well-known intrusions, so its accuracy can be guaranteed and thus its false alarm rate is relatively small in principle. However, because the normal behavior model in AID is built from the audit trails of the known normal behaviors, it is not possible to guarantee the completeness of the behavior model (c.f., Chapter 4). The knowledge scarcity about future behaviors is the main cause for false alarms in AID. In order to offset the knowledge scarcity, most AID techniques try to generalize the normal behavior model to cover more good behavior space [DDW00], and thereby to reduce false alarms. But model generalization tends to cover more intrusive behavior space as well, which degrades the detection rate. In other words, model generalization is two-edged for intrusion detection (c.f., Chapter 5).

Alternatively, anomaly context identification can be used to make up for the knowledge scarcity in the normal behavior model incrementally, thereby gradually reducing the false alarms. This is accomplished by including the anomaly contexts to augment the normal behavior model. It can be illustrated using the following example.

Example 9.7 *Suppose that there are two 3-grams in a test trace, which causes false alarms: ‘2-4-3’ and ‘3-1-1’ (i.e., anomaly contexts), and their frequency is 200 and 100 respectively. Without ACI, the number of false alarms is 300, but, with ACI, the two 3-grams will be inserted in the normal behavior model after an SSO’s inspection once their first (false) alarms are triggered. As a result, the number of false alarms will be reduced to 2 only.*

In the worst case where the SSO cannot perform the inspection in time, the ACI scheme is also useful. The alarms can be classified according to their anomaly contexts to reduce the number of alarms, and thus to lighten the inspection burden of the SSO.

Using this approach, similar to the intrusive behavior model in SID, the normal behavior model built from the audit trails of known normal behaviors can be guaranteed to be very accurate without the two-edged model generalization. Furthermore,

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

the normal behavior model can even start with zero-knowledge initially, and incrementally be built up using identified anomaly contexts as shown in the following example.

Example 9.8 *Take the same scenario as in example 9.4. Suppose that the test trace is ‘1-2-3-4’ but the normal behavior model is empty. Consequently, the first 3-gram ‘1-2-3’ will be detected as an alarm, and from the identified anomaly context ‘1-2-3’, it will be recognized as a false alarm, so ‘1-2-3’ is inserted into the normal behavior model. Then, if the second 3-gram ‘2-3-4’ triggers a false alarm as well, the normal behavior model will be augmented ultimately by two 3-grams: ‘1-2-3’ and ‘2-3-4’.*

The ACI scheme will make the normal behavior model cover more areas in the normal behavior space. On the other hand, the false alarms in AID techniques are caused by normal behaviors outside normal behavior model. Therefore, the false alarms can be suppressed by ACI in principle. In summary, the scheme will be useful to study the characteristics of intrusions, to remove false alarms, and hence to improve the efficiency of AID techniques. However, every coin has two sides. The identified anomaly contexts can also be utilized to create cleverer intrusions, such as the information hiding techniques [TMK02] and the mimicry attacks [WS02]. It is rightly said that the competition between the security researchers and the hackers is endless.

9.1.3 An IDS Architecture with ACI

Figure 9.1: A novel architecture of an intrusion detection system.

In general, an intrusion detection system consists of three functional components (Figure 9.1): the *event capturer* that extracts the utilized events from the stream of

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

audit trails; the *analysis engines* that identify signs of intrusions; and the *response components* that generate reactions based on the outcome of the analysis engines.

To utilize the advantage of the ACI scheme, an extra module is added into the architecture of an IDS as an offline module (Figure 9.1). Specifically, for every alarm produced by analysis engines, the ACI module will start to identify the anomaly context for determining whether it is a false alarm. In addition, the ACI module will take considerable amount of time to identify the anomaly contexts, so it is reasonable to propose it as an offline one.

9.1.3.1 The ACI Module

The architecture of the proposed ACI module is shown in Figure 9.2. There is a database for anomaly contexts, a graphical user interface (GUI) for an SSO, and four sub-modules for processing anomaly contexts.

Figure 9.2: The anomaly context identification module.

- **anomaly context database;**

It is the central component of the ACI module, in which alarms, the anomaly contexts and the inspection results of anomaly contexts by an SSO are stored for the task of anomaly context identification.

- **audit trails buffer;**

It stores the audit trails from the analysis engines with a specific context depth, which is defined as the size of the past context information in time remembered by ACI. It is worth noting that the audit trails in the buffer should include the anomaly contexts when an alarm occurs. For safety, the context depth should be large enough. However, the context depth should not be too large to affect the performance of the ACI module.

- **anomaly context mining;**

When an alarm occurs, its anomaly context will be mined from the audit trails buffer for processing by the next sub-module. The anomaly context is formatted with respect to a standard specification (similar to IDMEF[DCF05]).

- **anomaly context clustering;**

The mined anomaly context will be clustered and stored in the anomaly context database. If the anomaly context ever occurred in the database, it will be classified into the same cluster. Otherwise, a new cluster will be formed for the inspection of SSO. The new cluster may be combined with existing clusters after the inspection.

- **graphical user interface;**

It is a user-friendly interface to help SSO to inspect the anomaly contexts, and to manipulate the information related to anomaly contexts in the database. After inspection, a *decision* on whether the anomaly context is caused by false alarm(s) and its *confidence* level will be given. For convenience, the inspection result $\langle anomaly\ context, decision, confidence \rangle$ is named as an *ACI vector*.

- **anomaly context labeling;**

It is related to anomaly context clustering. If the SSO has inspected the cluster and there is an ACI vector of the cluster, the anomaly context will be labeled by the ACI vector. Otherwise, it will wait for the inspection results from SSO.

The workload of SSO may be very high if he inspects the anomaly contexts of all alarms. However, automatic preprocessing techniques for anomaly contexts can be proposed to alleviate the workload of SSO. For example, we can group these anomaly contexts if the anomaly contexts of alarms in different time instants are identical. Secondly, the anomaly contexts can be clustered, thus SSO does not need to inspect every anomaly context, and he only has to deal with a cluster of anomaly contexts. Furthermore, some practical experiences can also be applied in the preprocessing techniques. For instance, if the anomaly context, which causes the number of alarms to cross a limit within a period, is always from false alarms, then, we can automatically classify this anomaly context as a normal behavior. Nonetheless, the SSO is indispensable in our system as the task of SSO is to make up for the knowledge deficiency.

9.1.3.2 The Workflow of ACI.

The function of the ACI module is to collaborate with analysis engines to distinguish false alarms from true alarms, and to fine-tune the normal model in the analysis engines to avoid the same false alarms in the future. The procedures involved in ACI are as follows:

- (1) Every event processed by the analysis engines is copied to the ACI module to be stored in an audit trails buffer with a predefined context depth;
- (2) Each *alarm* generated by analysis engines is also sent to the ACI module;
- (3) The anomaly context of each alarm is mined from the audit trails buffer;
- (4) The anomaly context is clustered with any existing similar anomaly context, and stored into the anomaly context database. In addition, we can use some automatic techniques to preprocess the anomaly contexts.
- (5) Using GUI, an SSO inspects the anomaly context to determine its ACI vector. Both the anomaly context and its ACI vector are then stored in the anomaly context database.
- (6) The anomaly context is labelled with the ACI vector of its cluster in the anomaly context database;
- (7) The ACI vector of the alarm is sent to analysis engines and/or response components to trigger response strategies;
- (8) The analysis engines fine-tune the normal model using the ACI vector to avoid the same false alarms in the future.
- (9) The response components produce responses using the ACI vector.

In general, the root cause analysis in [Jul03] can be applied here to simplify the work in the ACI module, in which the anomaly contexts for alarms can be clustered to avoid the repeatable work by the SSO. At the same time, the logic in alert correlation [NCRX04] can also be applied here to rebuild (normal/intrusive) scenarios. In addition, the ACI vectors and corresponding anomaly contexts are stored in the anomaly context database for future reference, and the database is similar to the intrusion signature set in SID.

9.1.4 Other uses of the ACI scheme

Other than the main objective of reducing false alarms, the ACI module can be utilized as an analysis tool for a number of tasks as listed below.

- to dig out the intrusion signatures for signature-based IDS since the SID system is computationally more efficient than the AID system.
- to strengthen alert correlation for AID techniques since one alert cannot reliably determine an intrusion, and its anomaly context can give the exact reasons.
- to study the intrusive characteristics of intrusions, and then to improve the efficiency of existing AID techniques, and to propose more efficient AID techniques.
- to apply it into other anti-intrusion fields, such as computer forensics.

In the following sections, we will show how to mine anomaly contexts of alarms from audit trails in this thesis. A simple but representative ACI scheme is given in the following sections. First, using the theoretical basis behind an AID technique, we define what is anomaly context of an alarm and what is minimal anomaly context in the identified anomaly context. Then, we design a mechanism to show how to identify the anomaly contexts of alarms from the audit trails.

9.2 Anomaly Contexts in stide: A Case Study

As mentioned above, our treatment of anomaly context in this case study is based on stide. For succinctness, we introduce a formalism here to discuss and analyze the anomaly contexts of stide, as well as to describe the specific ACI implementation.

9.2.1 Notations and Definitions

Sequences and Sequence Sets: Let Σ denote the dataset for a process, which consists of event logs with the identity of the associated running process. $SS(\Sigma, l)$ denotes the set of all the sequences of length l ($l \geq 0$), which are collected from Σ . As a special case, $SS(\Sigma, 0) = \{\phi\}$. Furthermore, $SS(\Sigma) = \bigcup_{l=0}^{+\infty} SS(\Sigma, l)$, and S is a sequence in $SS(\Sigma)$.

Set Operations: For given datasets Σ_a and Σ_b of a process and corresponding sequence sets $SS(\Sigma_a, l)$ and $SS(\Sigma_b, l)$ ($l \geq 0$), the set operations ($\cup, -$) are defined as

follows:

- (1) $SS(\Sigma_a, l) \cup SS(\Sigma_b, l) = \{S | (S \in SS(\Sigma_a, l)) \vee (S \in SS(\Sigma_b, l))\}$
- (2) $SS(\Sigma_a, l) - SS(\Sigma_b, l) = \{S | (S \in SS(\Sigma_a, l)) \wedge (S \notin SS(\Sigma_b, l))\}$

Foreign Sequences and Self Sequences. Let Σ_a be the *training dataset*, and Σ_b be the *test dataset*. For any sequence $S \in SS(\Sigma_b)$, if S is also in $SS(\Sigma_a)$, S will be called a *self sequence*, otherwise, it is a *foreign sequence* to Σ_a . For the convenience of expression, $FRGN(\Sigma_b|\Sigma_a)$ is defined as the set of foreign sequences of Σ_b w.r.t. Σ_a . Similarly, the set of self sequences is defined as $SELF(\Sigma_b|\Sigma_a)$. Obviously, $\phi \in SELF(\Sigma_b|\Sigma_a)$, and $FRGN(\Sigma_b|\Sigma_a) \cup SELF(\Sigma_b|\Sigma_a) = SS(\Sigma_b)$.

A sequence S in $FRGN(\Sigma_b|\Sigma_a)$ will be called a *minimum foreign sequence (MFS)* [TM03] if none of its subsequences² is in $FRGN(\Sigma_b|\Sigma_a)$, i.e. all of its subsequences are in $SELF(\Sigma_b|\Sigma_a)$. The set of all minimum foreign sequences is denoted as $MFS(\Sigma_b|\Sigma_a)$. and the smallest length of all its sequences is defined as:

$$|MFS|_{min}(\Sigma_b|\Sigma_a) = \min_{S \in MFS(\Sigma_b|\Sigma_a)} length(S) = \min(\{l | l \geq 0; SS(\Sigma_b, l) - SS(\Sigma_a, l) \neq \Phi\})$$

where, for convenience, we use the notation $|\dots|$ to represent the length of any member sequence in the sequence set MFS, instead of the size of the set. In words, $|MFS|_{min}(\Sigma_b|\Sigma_a)$ represents the minimum length of any sequence $S \in \Sigma_b$ that is not a sequence in Σ_a . From this definition, $|MFS|_{min}(\Sigma_b|\Sigma_a) \geq 1$.

9.2.2 Anomaly Contexts in stide

In the experimental setup for stide [WFP99], there are two datasets for every process, the normal dataset Σ_{nml} , and the intrusive dataset Σ_{int} . A normal dataset is a dataset that must be collected in the normal run of the process without any intrusion, and an intrusive dataset Σ_{int} was collected when one or more intrusions were occurring targeting the process.

In terms of these two datasets from the same process, stide can be described as follows. Let $\omega (\geq 1)$ denote the size of the detector window. In the modeling phase, the normal model of the process is obtained as: $SS(\Sigma_{nml}, \omega)$. Then, in the detection phase, the foreign sequences in the intrusive dataset Σ_{int} , $FS(\Sigma_{int}|\Sigma_{nml}, \omega)$, are enumerated:

$$FS(\Sigma_{int}|\Sigma_{nml}, \omega) = SS(\Sigma_{int}, \omega) - SS(\Sigma_{nml}, \omega) \quad (\text{Eq. 9.1})$$

²Subsequence and supersequence will always imply contiguity in this chapter.

If $FS(\Sigma_{int}|\Sigma_{nml}, \omega) \neq \Phi$, the intrusion(s) in the intrusive dataset Σ_{int} can be detected with the detector length ω [HFS98, TM03, WFP99]. Thus, any sequence S in $FS(\Sigma_{int}|\Sigma_{nml}, \omega)$ will cause an alarm in stide, and S is its *anomaly context*. It is evident that the sequence set $FS(\Sigma_{int}|\Sigma_{nml}, \omega)$ is strongly related to $MFS(\Sigma_{int}|\Sigma_{nml})$ via $|MFS|_{min}(\Sigma_{int}|\Sigma_{nml})$:

$$|MFS|_{min}(\Sigma_{int}|\Sigma_{nml}) \leq \omega \Leftrightarrow FS(\Sigma_{int}|\Sigma_{nml}, \omega) \neq \Phi \quad (\text{Eq. 9.2})$$

Practically, even though the underlying principle is very simple, stide can detect most intrusions into the processes (*in the datasets from UNM* [HFS98]). For this reason, it is accepted as a typical AID technique. It is obvious that any sequence S in $MFS(\Sigma_{int}|\Sigma_{nml})$ will cause an alarm at least using a proper detector window ω , and S is the minimal anomaly context of the alarm as any curtailment of S (which implies loss of context information) will cause it to fall into $SELF(\Sigma_{int}|\Sigma_{nml})$ (i.e. no alarms will occur).

9.2.3 Mining Anomaly Contexts from Audit Trails

From the above description, we know that anomalies in the intrusive dataset will be indicated by a foreign sequence set $FS(\Sigma_{int}|\Sigma_{nml}, \omega)$. Therefore, it will be helpful to visualize these foreign sequences first in our ACI scheme. Furthermore, the detector window size ω is also critical to the foreign sequence set because the foreign sequence will not be detected if its length is larger than ω . Therefore, in the graphs for the visualization of anomaly contexts, the length of every foreign sequence in the intrusive dataset will be shown to indicate whether it can be detected by the stide detector with length ω . The graph shows the precedent foreign sequence length of the current event (vertical) against its index (horizontal). For convenience, we call these graphs as *foreign sequence graphs* (**FSG**). To identify anomaly contexts correctly and precisely, we identify every MFS sequence, and the events associated with the MFS sequence can then be identified as the minimal anomaly context.

To achieve it, the intrusive dataset can be processed in two ways: either (1) splitting it into blocks, and to evaluate every block; or (2) evaluating every event in it. The first method of splitting will have the possibility of breaking the foreign sequences. Therefore, we will use the second approach, i.e. every event in the intrusive dataset will be evaluated for its significance in terms of anomaly context identification.

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

9.2.3.1 FSG: Foreign Sequence Graph

An FSG graph is produced as follows. Suppose that the audit trails of a process is $\{e_1, e_2, \dots, e_p\}$, and the lengths of stide detectors range from 1 to N . First, the stide detectors are built from the normal dataset with the detector window size ω varying from 1 to N . Then, for every event e_i , if $e_{i-l+2} \dots e_i \in SELF(\Sigma_{in}|\Sigma_{no})$ but $e_{i-l+1} \dots e_i \in FRGN(\Sigma_{in}|\Sigma_{no})$, the length of the precedent foreign sequence $\ell_i = length(e_{i-l+1} \dots e_i) = l$, named as foreign sequence length at the event e_i , will be shown in the FSG graph. Specifically, the following algorithm can be used to calculate the foreign sequence length series. To illustrate the anomaly context, the detection results

Algorithm 7 Calculating the foreign sequence length.

Require: The event sequence of the process $e_1, e_2, \dots, e_p$; and the stide self model series $stide_1, stide_2, \dots, stide_N$;

```

1: for i=1 to p do
2:   seq=NULL;  $\ell_i = N + 1$ ;
3:   for j=0 to N-1 do
4:     If  $i-j \leq 0$  break;
5:     Insert  $e_{i-j}$  to seq as the first element;
6:     if seq is-not-in  $stide_{j+1}$  then
7:        $\ell_i = length(seq)$ ; break;
8:     end if
9:   end for
10: end for

```

Output $\ell_1, \ell_2, \dots, \ell_p$

$\ell_1, \ell_2, \dots, \ell_p$ are used to draw the foreign sequence graph for the intrusive process.

Example 9.9 Suppose that the training audit trails trace is ‘A-B-C-B-A’, $N = 5$, and the test trace is ‘A-B-A-B-C-A’. The stide self model is: $stide_1 = \{A, B, C\}$, $stide_2 = \{A-B, B-C, C-B, B-A\}$, $stide_3 = \{A-B-C, B-C-B, C-B-A\}$, $stide_4 = \{A-B-C-B, B-C-B-A\}$, and $stide_5 = \{A-B-C-B-A\}$. Using the algorithm 7, we will obtain, $\{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5, \ell_6\} = \{6, 6, 3, 3, 4, 2\}$ ($p = 6$). From these values, we know that, for instance, the third event in the test trace (which is ‘A’) is associated with a foreign sequence of minimum length 3 (since $\ell_3 = 3$). This conforms with the normal model because the smaller sequences ending in ‘A’, namely ‘A’ and ‘B-A’, are both found in the normal model, but the sequence ‘A-B-A’ is not.

An anomaly context will be determined from the FSG when the events near it have smaller foreign sequence length values compared with its neighbours. In practice, if one

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

minimum foreign sequence is $e_me_{m+1} \dots e_n$ ($n - m + 1 \leq N$), it will be identified by the local lowest point in its foreign sequence graph, which is expressed as $\ell_n = n - m + 1$, since $\ell_{n-1} \geq n - m + 1$, and $\ell_{n+1} \geq n - m + 1$. Thus, the precedent $n - m + 1$ events consist of the minimal anomaly context, that is, the MFS sequence.

9.2.3.2 MFSs Identified from FSGs

According to the definition of minimum foreign sequences, the non-MFS foreign sequences contain one or more minimum foreign sequences. For example, suppose that one minimum foreign sequence is $x_1x_2 \dots x_l$, the foreign sequence can be constituted as follows: $y_1 \dots y_mx_1x_2 \dots x_ly_1 \dots y_n$, where, $m \geq 0$, $n \geq 0$, $y_1 \dots y_m$ and $y_1 \dots y_n$ are non-MFS sequences. Because the non-MFS foreign sequences provide no additional information in comparison to the MFS(s) in them [TM03], they will be filtered out.

Fortunately, in the generation of the foreign sequence graph (Algorithm.7), the prefix sequence $y_1 \dots y_m$ is filtered out in the beginning. Therefore, to collect the minimum foreign sequence, only the suffix sequence $y_1 \dots y_n$ should be filtered out. Therefore, from the foreign sequence graphs, the method for trimming down non-MFS foreign sequences can be inferred as follows: if $\ell_i = \ell_{i-1} + 1$, the foreign sequence identified by ℓ_i will be filtered out since it is included in the foreign sequence identified by ℓ_{i-1} .

Example 9.10 *Take the same scenario as example 9.9. The minimum foreign sequences identified from the example test trace are $\{A-B-A, B-A-B, C-A\}$. Because $\ell_5 = \ell_4 + 1$, the foreign sequence ‘B-A-B-C’ is filtered out, i.e., it is not a minimal foreign sequence.*

9.3 Evaluations

The datasets from the University of New Mexico [TM03, WFP99] have been used in our experiments. The normal and intrusive datasets for selected processes are specified in Table 9.1. From the table, our selected datasets represent most processes and intrusions. In order to analyze the characteristics of every intrusion, its intrusive dataset, even into the same process, is treated as an individual dataset since every intrusion will leave its own characteristic anomaly contexts in its audit trails.

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

Normal Datasets	Intrusive Datasets	No. of Traces	No. of System Calls
live-named-UNM	—	142	9230572
	buffer overflow-1	3	969
	buffer overflow-2	2	831
live-lpr-MIT	—	2703	2926304
	lprcp	1001	165248
sendmail-CERT	—	294	1576086
	syslog-local-1	6	1516
	syslog-local-2	6	1574
	syslog-remote-1	7	1861
	syslog-remote-2	4	1553
	cert-sm565a	3	275
	cert-sm5x	8	1537
sendmail-UNM	—	346	1799764
	decode	36	3067
	forward loops	36	2569
	sunsendmailcp	3	1119
syn-wu-ftpd	—	8	180315
	misconfiguration	5	1363
syn-xlock-UNM	—	71	339177
	buffer overflow-1	1	489
	buffer overflow-2	1	460

Table 9.1: The dataset specifications.

9.3.1 Visualizing Anomaly Contexts

Figure 9.3 illustrates the foreign sequence graphs for ‘named’ and two intrusion instances ‘buffer overflow’. For the convenience of comparison, two FSG graphs are compressed into this figure, and their borders are split with vertical lines for different intrusive datasets. To illustrate the boundaries, we introduce dummy values of $\ell=-4$ between different intrusive datasets, and $\ell=-1$ between different processes in one dataset. For instance, the vertical line at ‘974’ splits the two intrusive datasets for ‘buffer overflow’ into ‘named’. The maximum detector window length N is fixed at 25 in our experiments, so the maximum value of the foreign sequence length is 26 (Algorithm 7). Every point in the FSGs is the foreign sequence length corresponding to the index of a system call (in the horizontal axis).

One significant phenomenon reflected in both FSG graphs is that the intrusions will not cause foreign sequences in the initial stage. Only after 600 system calls, there are some foreign sequences generated. To some degree, this indicates that there is a ‘warmup’ stage for most intrusions, at least for the intrusions detected by stide-like AID detectors. This phenomenon is common for most processes provided that most FSGs in Figure 9.4 are consistent to have such a ‘warmup’ stage.

Figure 9.3: The foreign sequence graphs for process ‘named’ and intrusion ‘buffer overflow’.

FSGs for All Processes. Figure 9.4 summarizes all foreign sequence graphs in our experiments. Among these FSG graphs, the number of foreign sequences³ is so significant that most intrusions can be detected by stide.

Three instances of the intrusion ‘sunsendmail’ in subfigure (g) have caused identical FSG graphs. That is to say, they have the same intrusion characteristics. At the same time, for the different instances of the same intrusion in subfigures (a)(c)(f) or (i), their FSG graphs are very similar as well. This suggests that it is unnecessary to design specific technique to detect different runs of an intrusion.

For different intrusions into a process, their FSG graphs are different. For example, for the process ‘sendmail’ from CERT, the FSG graph of the intrusion `syslog` in subfigure (c) is different from the FSG graph of the intrusion `sm565a` or `sm5x` in subfigure (d). Similarly, the FSG graphs of different intrusions into the same process ‘sendmail’ from UNM are different as well as can be seen from subfigure (e) (for intrusion ‘decode’), subfigure (f) (for intrusion ‘forward loops’) and subfigure (g) (for intrusion ‘sunsendmailcp’). This fact proves that there are no general detection strategies to detect all intrusions into a process, and specific detection strategies (e.g. stide detectors with different length) are needed to detect different intrusions.

There are precursor foreign sequences with larger length in the FSG graphs of some intrusions, especially in subfigures (c)(d)(e)(f) and (g). Therefore, under these

³Actually, the foreign sequences are anomaly contexts in this case study.

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

9.4.a: Process 'named'.

9.4.b: Process 'lpr'.

9.4.c: Process 'sendmail-CERT'.

9.4.d: Process 'sendmail-CERT'.

9.4.e: Process 'sendmail-UNM'.

9.4.f: Process 'sendmail-UNM'.

9.4.g: Process 'sendmail-UNM'.

9.4.h: Process 'ftpd'.

9.4.i: Process 'xlock'.

Figure 9.4: Foreign sequence graphs. The intrusions are: (a) 'buffer overflow'; (b) 'lprcp'; (c) 'syslog'; (d) 'sm56a' and 'sm5x'; (e) 'decode'; (f) 'forward loops'; (g) 'sunsendmailcp'; (h) 'misconfiguration'; (i) 'buffer overflow'.

scenarios, the larger the stide detector size is, the sooner the first alarm will be raised to indicate the occurrence of an intrusion. However, it is well known that larger stide detector windows will degrade the efficiency of stide. That is, there is a clear tradeoff between the MTTA (Mean Time To Alarm) and the efficiency of stide: if we want to detect the intrusion earlier (thereby lowering the MTTA), we need a larger detector window, degrading the detection efficiency.

9.3.2 Identifying Minimal Anomaly Contexts

The minimum foreign sequences in the audit trails are characteristics left by an intrusion, which is utilized by stide-like anomaly detectors to detect the intrusion. As mentioned earlier, MFSs are the minimal anomaly contexts of alarms in stide. Based

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

on the identified anomaly contexts in these datasets, their numerical statistics are summarized in Figure 9.5 and Table 9.2. Note that the shared MFSs in Table 9.2 are the stable anomaly contexts of their corresponding intrusions.

9.3.2.1 Numerical Statistics of Identified MFSs

MFS Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	sum
named-1	5	34	11	0	3	1	0	2	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	59
named-2	5	19	6	1	1	1	0	1	0	0	2	0	1	0	1	0	0	0	0	0	0	0	0	0	0	38
lpr-1	26	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	37
sm-cert-1	3	20	13	5	1	4	2	0	0	1	0	0	1	1	0	0	2	2	0	0	0	0	0	0	0	55
sm-cert-2	3	27	20	5	2	3	4	0	0	1	0	0	1	1	0	0	2	2	0	0	0	0	0	0	0	71
sm-cert-3	3	30	20	5	2	3	4	2	0	1	1	0	2	1	0	0	2	2	0	0	0	0	0	0	0	78
sm-cert-4	2	9	9	4	1	3	4	2	0	1	1	0	2	0	0	0	2	2	0	0	0	0	0	0	0	42
sm-cert-5	0	8	2	2	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	15
sm-cert-6	0	23	25	17	6	4	5	2	2	1	0	0	1	0	0	0	4	2	4	0	0	0	0	0	0	96
sm-unm-1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2
sm-unm-2	0	2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	5
sm-unm-3	0	11	9	5	2	0	2	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	33
sm-unm-4	0	3	1	2	0	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	11
sm-unm-5	0	11	9	2	2	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	30
sm-unm-6	0	10	10	5	2	0	2	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	33
sm-unm-7	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
sm-unm-8	0	7	7	4	2	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	24
sm-unm-9	0	7	7	4	2	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	24
sm-unm-10	0	7	7	4	2	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	24
ftpd-1	3	34	14	8	4	2	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	68
xlock-1	10	41	8	2	2	3	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	68
xlock-2	10	41	8	2	2	3	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	68

named-1,2: the buffer overflow-1,2

sm-cert-1..6: syslog-local-1,2, syslog-local-3,4 sm565a and sm5x

sm-unm-1..10: decode-280,314, forward loops-1,2,3,4,5, sunsendmailcp-10763,10801,10814

ftpd-1: misconfiguration

xlock-1,2: buffer overflow-1,2

Figure 9.5: The number of MFS sequences (non-duplicated) in all intrusive datasets.

From the total number of non-duplicated MFS sequences in every intrusive dataset (Figure 9.5), we can infer that (1) The detector window $\omega = 2$ is enough to detect most of the intrusions except the intrusion instance ‘decode-280’; (2) From the FSG in Figure 9.4(e) and the numerical statistics in Figure 9.5 of the intrusion instance ‘decode-280’, we observe that the minimum length of MFSs is 6. From its positional index, we can easily find that this corresponds to the sequence ‘2-95-6-6-95-5’. The following list shows all the MFSs occurring in the two decode instances and the corresponding system calls:

- *decode-280*
process 283: 2-95-6-6-95-5
- *decode-314*
process 317: 112-6, 6-19, 2-95-1, 2-95-6-6-95-5.

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

*1:exit, 2:fork, 5:open, 6:close, 19:lseek, 95:connect, 112:vtrace

From this observation and Eqn (3), we can now confidently assert that the intrusion instance ‘decode-280’ leads to the magic number 6 for stide [WFP99, TM03] since $|MFS|_{min}(\Sigma_{int}|\Sigma_{nml}) = 6$, and (3) the stable anomaly context for ‘decode’ is ‘2-95-6-6-95-5’ and other anomaly contexts are transient. Most importantly, the number of stable anomaly contexts of most intrusions (except forward loops) are large enough to characteristic the intrusion. (4) As there are only a small number of minimum foreign sequences with their lengths beyond 7, the efficiency of stide detectors will not be improved much if the detection window $\omega > 7$.

Intrusion	Num. of Runs	MFSs per Run	Num. of Shared MFSs
decode	2	{2,5}	2
buffer overflow into xlock	2	{68,68}	68
buffer overflow into named	2	{59,38}	33
sunsendmailcp	3	{24,24,24}	24
forward loops	5	{33,11,30,33,4}	0
syslog-local	2	{55,71}	52
syslog-remote	2	{78,42}	42

Table 9.2: Shared MFS sequences of the same intrusion into the same process.

In addition, we also note that different runs of an intrusion will share most MFS sequences (Table 9.2), and these shared MFS sequences among several intrusion runs (i.e., *stable anomaly contexts*) can be utilized to study the general intrusion characteristics (by an SSO). Especially, different runs for ‘sunsendmailcp’ have the same set of minimum foreign sequences. This discovery benefits the research on AID techniques because the diversity of different runs of the same intrusion is not too large to warrant the designing of one specific (or ad-hoc) IDS system for each of its runs. At the same time, it also strengthens our earlier assertions that the FSG graphs are intrusion-specific, and that different runs of an intrusion almost have the same intrusion characteristics, i.e. the anomaly contexts. Furthermore, the existing of non-shared MFSs for different runs of an intrusion show the existing of ‘*transient*’ anomaly contexts under different conditions/environments.

Also, from Figure 9.5 and Table 9.2, one can see that the large number of minimum foreign sequences, especially for the intrusion ‘buffer overflow’, will discourage the mimicry attacks [WS02] and the information hiding paradigm [TMK02] greatly. This is because, to make the intrusions transparent to the detector, all the minimum foreign sequences should be mimicked before these attacks can be launched with success.

9.4 Further Discussions

In principle, we should define what is the (*minimal*) anomaly context of an alarm using the basic foundations of intrusion detection techniques. For example, the minimal anomaly context in stide is the minimal foreign sequence in the test trace. However, not all AID techniques can be formalized, and the minimal anomaly context cannot always be defined. Thus, a practical definition for the anomaly context of an alarm becomes critical for our proposed ACI module in the new IDS architecture.

In practice, we can estimate the anomaly contexts from the audit trails as follows. Suppose that an audit trail for intrusion detection is denoted as Σ . In general, we collect a feature vector instance I from Σ in each time instant [LDZ05c]. Ultimately, a series of instances, $I_1, I_2, \dots, I_p$, is built from the audit trails, where the instance I_i is collected in the i -th time instant. To determine whether the instance I_m ($m \leq p$) is left by an intrusion, a series of instances around I_m , $I_x \dots I_m \dots I_y$, is needed in the detection phase ($x \leq m \leq y$). Thus, if an alarm is triggered by the instance I_m , the series of instances in the audit trails (that is, $I_x \dots I_m \dots I_y$) is defined as an anomaly context of the alarm, i.e., the anomaly context of the alarm is $I_x \dots I_m \dots I_y$. In such case, the context depth of the *audit trails buffer* in ACI module is $y - x + 1$. For example, for stide [HFS98] with detector window $w = 6$, $y = m$ and $m - x = 5$, the anomaly context is $I_{m-5} \dots I_m$.

Using such estimated anomaly contexts for alarms, a general ACI scheme can be built to suppress the false alarms, and to fine-tune the behavior models. Furthermore, the ‘*stable*’ anomaly contexts for a number of intrusion runs can be utilized to extract the signature for the intrusion. At the same time, the ‘*transient*’ anomaly contexts should be treated specially by excluding it from normal/intrusive behavior models. Nonetheless, an exact definition for anomaly contexts is needed with respect to specific AID techniques. In addition, it is worth noting that the ACI module should be flexible enough to tackle the wrong ACI vectors for some anomaly contexts. This is because there are some unavoidable mistakes in the inspection results from SSO and automatic preprocessing techniques.

9.4.1 Put it Into Our Box: ACI with USAID

Within the above case study for ACI, every system call is a state in the execution of a process, and the context information between system calls is utilized to constitute the

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

anomaly contexts of alarms. In our proposed framework (i.e., USAID in Chapter 3), every signature could be regarded as a resource state at a time instant, and hence is equivalent to a ‘*system call*’ discussed in this chapter. Furthermore, the instance I_m in our above discussion is similar to a behavior signature in the outputs of automatic signature assemblers in Chapter 8. Therefore, the context information between signatures will be further utilized in ACI for intrusion detection. Figure 9.6 gives the integral intrusion detection system after combining automatic signature assemblers, USAID or varUSAID, and anomaly context identification modules that we proposed in our thesis. We are developing such a system to further validate our proposed solutions.

Figure 9.6: An integral intrusion detection system with three functional modules.

The integral intrusion detection system is flexible enough to detect intrusions in both host-based and network-based audit trails. Within host-based applications, we have validated the ACI scheme in this chapter by using the system calls of a process. More comprehensive applications need to consider various parameters along with system calls in building behavior signatures. For network-based applications, it may be better to deploy the ACI scheme in a session-based approach rather than a packet-based approach. This is because the context information among packets within a session is more tight than that among packets across several sessions, and it is utilized in defining anomaly contexts of alarms.

9.5 Summary

In this chapter, the concept of anomaly context identification was introduced to suppress the high false alarm rate in AID technique. A simple but representative prototype of the concept was proposed based on a typical AID technique, stide, in which

CHAPTER 9. ANOMALY CONTEXT IDENTIFICATION

the minimum foreign sequences in the intrusive datasets were identified as anomaly contexts. The experimental results shown that the anomaly contexts can be identified from the audit trails successfully, and the intrusion characteristics can be studied from the stable anomaly contexts of an intrusion. One future work is to incorporate the anomaly context identification scheme into our proposed framework for intrusion detection, and to validate it in further applications.

Chapter 10

Conclusions and Outlook

In this chapter, we summarize the work presented in this dissertation, review the main contributions made, and discuss possible work for future research in our direction.

10.1 Summary

This thesis consists of two main parts: *problem analysis* and *solutions* for intrusion detection. In order to achieve them, it proposed a comprehensive framework for intrusion detection. Under this framework, the generic problems for intrusion detection were analyzed theoretically, and their root cause, model generalization, was evaluated experimentally to identify its implications for intrusion detection. Inspired by the analysis and evaluation results, a novel intrusion detection technique and several practical solutions for generic problems were developed. Several case studies were also conducted to validate our proposals.

Problem Analysis. We first motivated our thesis by emphasizing the importance of intrusion detection in the defense of increasing vulnerabilities and threats: the vulnerabilities are increasing with more Internet applications, and the situation will persist due to the flawed software development. We discussed the common knowledge on intrusion detection techniques: their main components and their evolution history. We also summarized the generic problems in intrusion detection. We explained that, concurring with McHugh for further progress in intrusion detection, it is necessary to analyze these generic problems for their root causes, and/or to propose solutions to these problems. To this end, a formal framework for intrusion detection is needed.

Then, we reviewed and discussed existing frameworks for intrusion detection and found that they lack the capability to analyze these problems. Mimicking the object

CHAPTER 10. CONCLUSIONS AND OUTLOOK

identification process in human beings, we proposed a novel framework for intrusion detection. We practically explained the reasonableness of two necessities in our framework, namely, *the training audit trails* and *a general feature vector*, and described the framework in detail using several new concepts, namely, **NSA label**, **feature range**, **compound feature**. Promisingly, the intrusion detection problem is formally defined in this framework. We finally enumerated the promising properties of our framework, which are used in a number of applications discussed below.

As the first application of our framework, we analyzed the generic problems in SID and AID techniques. Specifically, two quality factors in behavior models are identified to interpret the generic problems in a quantified manner. We further discussed the unanswered questions in intrusion detection, and then proposed a practical method to answer them. We argued that model generalization is the (even only) component that affects the quality factors in behavior models. For this reason, a multi-level layered evaluation framework was designed to identify the implications of model generalization on intrusion detection. In our case study, the significance of every level of generalization is identified successfully from the experimental results. Although we only validated the evaluation framework for a typical dataset, our procedure is general enough to be applied to evaluate model generalization using other datasets.

Practical Solutions. We designed a novel intrusion detection technique, USAID, using the framework and the analysis results achieved. Unlike the two existing intrusion detection approaches, which use only partial knowledge of a resource, USAID utilized all of its available knowledge. Its detection performance was promising with respect to detection rate and false alarm rate. Significantly, the anomalies in the output of USAID were categorized with different false alarm rates. We then proposed varUSAID to reduce the intensive computation in the detection phase. In varUSAID, feature ranking and variable-length signatures worked collaboratively. First, the features with overlapping distinguishability were discarded. Then, the variable-length signature was utilized to further reduce the length of a signature. However, our experimental results showed that information discarded in the process is often useful for intrusion detection. Specifically, the seemingly redundant information discarded by variable-length signatures is useful in detecting novel intrusions.

In order to address the problems faced by manually designed intrusion signatures, we devised a mechanism to automatically generate intrusion signatures. We utilized

the formal definition of behavior signatures to design a signature assembly line. To make the signature assembly process automatic, we needed to discover the feature ranges of every feature in advance. Therefore, we evaluated several methods to mine the feature ranges from training audit trails. A lot of future work is needed to propose post-processing techniques for the outputted signatures from the assembly line.

We also proposed a practical scheme to suppress false alarms in AID and USAID techniques, namely ACI or Anomaly Context Identification. As an effective technique to make up for the inadequacy of knowledge for determining whether an anomaly is caused by intrusion(s), it lets security site officers to make up for such knowledge deficiency on the fly by studying the contexts of anomalies (i.e., alarms). Experiments have shown that the (minimal) anomaly contexts can be mined from the audit trails successfully. Our future work will include studying how to incorporate our proposed solutions (i.e., *USAID/varUSAID*, *automatic signature assemblers*, *anomaly context identification*) into an integral intrusion detection system.

10.2 Thesis Contributions

We list the thesis contributions here.

- **Proposing a comprehensive framework for intrusion detection.** We mimicked the object identification process in human beings for intrusion identification in the framework, in which every feature is made discrete using a new concept of feature range. We studied the reasonableness of two critical assumptions in the framework, and stated that most past research on intrusion detection follow the same two assumptions. We then enumerated its attractive properties for intrusion detection, and its applications using these properties.
- **Analyzing generic problems in intrusion detection.** We first identified the two main quality factors in behavior models from the relations between behavior spaces and models. We then quantified the behavior models, spaces and two quality factors for problem analysis using the number of signatures. We related the detection performance to the generic problems and interpreted them reasonably in a quantified manner. Ultimately, we identified their root cause.

- **Evaluating model generalization for its implications on intrusion detection.** We designed a multi-level framework to evaluate the effects of model generalization. In every level, there are at least one inference assumption, and all inference assumptions were sorted based on their reasonableness. Specifically, within our proposed intrusion detection framework, there are three levels of model generalization. We evaluated model generalization level by level and identified their implications for intrusion detection experimentally.
- **Designing new intrusion detection techniques, USAID/varUSAID.** We applied our intrusion detection framework to propose new intrusion detection techniques as well. The above analysis results were utilized in designing their architecture. In varUSAID, we sorted features in a feature vector, and discarded the features with overlapping distinguishability. We realized that the model redundancy reduced by variable-length signatures is useful for intrusion detection, especially for anomaly detection. We showed that USAID/varUSAID have promising detection performance for both known and novel intrusions. Significantly, we categorized the outputted anomalies from USAID successfully by showing different false alarm rates.
- **Presenting an automatic signature assembly line.** Most intrusion signatures were manually constructed due to the lack of a formal definition for behavior signatures. To this end, we utilized the formal definition for behavior signatures in our framework to design a signature assembly line. Lastly, we examined several methods for discovering the signature materials (i.e., feature ranges) for the assembly line.
- **Identifying anomaly contexts of alarms to suppress false alarms.** Knowledge deficiency about anomalies in AID techniques leads to higher false alarm rate. To make up for the knowledge deficiency, we proposed a new mechanism to suppress the false alarms, in which anomaly context of every alarm is utilized to identify whether the alarm is false. We shown that the anomaly contexts of alarms can be mined from audit trails successfully. As an initial work, we are trying to put the mechanism into our proposed framework.

10.3 Future Work

As stated in our thesis, our proposed intrusion detection framework has attractive properties. We have applied them to propose several solutions for the generic problems. There also remain several interesting and important future directions:

- IMPROVED SOLUTIONS FOR GENERIC PROBLEMS.

We have designed several solutions for the generic problems based on our analysis results. However, further research is needed to improve on these solutions to eliminate or mitigate these problems of intrusion detection. For example, [XDC04, SY05] have proposed some methods to prevent mimicry attacks in AID techniques under some scenarios. These two methods are consistent with our analysis result in Chapter 4 that mimicry attacks can be mitigated by using more features from the audit trails. Alternatively, like what we have done in this thesis, we can analyze generic problems more comprehensively by developing a more comprehensive framework for intrusion detection. In addition, it is possible that one solution will alleviate one problem but aggravate some other problem. An efficient solution should carefully consider the tradeoffs involved.

- METHODOLOGIES AND TOOLS FOR FEATURE EXTRACTION FROM AUDIT TRAILS.

In our framework for intrusion detection, we assume that there exists a predefined feature vector. In fact, most existing intrusion detection techniques utilize a predefined feature vector implicitly as well. Therefore, it is necessary to extract the critical features from audit trails for intrusion detection. We can first collect all features, which have been utilized or can be utilized, within the audit trails. Then, feature ranking and selection are applied to get the significance of every feature, and some features with overlapping distinguishing capability can be discarded. For example, suppose that the distinguishing capability of features F_a , F_b and F_c includes that of features F_d and F_e . Thus, if F_a , F_b and F_c are applied in intrusion detection, there is no need to use F_d and F_e . We also need to build tools to extract the significant features from audit trails efficiently. In addition, it will benefit intrusion detection if we can identify the evolution patterns for features along with intrusion evolution.

CHAPTER 10. CONCLUSIONS AND OUTLOOK

- EVALUATIONS OF MODEL GENERALIZATION ON OTHER DATASETS.

Due to the incompleteness of training audit trails and insufficient distinguishing capability of used features, model generalization is widely used to infer the status of new instance in the unknown behavior subspaces. Using our proposed evaluation framework, we can further evaluate it on other datasets. Considering that some techniques have their own specific application environments in intrusion detection[Axe00a, DDW00], it is possible that its implications on intrusion detection are different under different application scenarios and environments. Furthermore, the evaluation framework can also be utilized to identify the usefulness of model generalization on other research areas, such as bioinformatics, outlier detection.

- ENSURING QUALITY OF THE TRAINING AUDIT TRAILS.

In this thesis, we implicitly assume that the training dataset is always correct. However, it is possible that there are some errors in the labels or feature values. In other words, there is a problem involving the quality of the training audit trails. One future work is to evaluate the influence (or sensitivity) of the quality of the training audit trails on the quality of the behavior models. Because some novel intrusions could be mixed in collecting the training audit trails in real deployment, this evaluation can predict the detection performance of the behavior models from the training audit trails more exactly.

- INTRUSION DETECTION EVALUATION.

In our analysis, we have identified the significance of various levels of model generalization in our framework. By adjusting the model generalization, we can synthesize artificial datasets with a specific extent of generalization. Existing intrusion detection techniques are then evaluated using the artificial datasets. Unlike MIT Lincoln lab 1998/1999 IDS evaluation [HLF⁺01, Ken99] where all intrusions are known before dataset generation, we can evaluate these techniques without regard to known intrusions. Furthermore, considering the intrusion evolution and the extent of generalization needed to detect an existing intrusion, we can infer what is the generic detection performance of a technique.

- POST-PROCESSING TECHNIQUES FOR THE SIGNATURE ASSEMBLY LINE.

The outputted signatures from the signature assembly line (in Chapter 8) need

further processing. The post-processing techniques are to process them, and thereafter to determine their anomalousness. Example of postprocessing include the signature correlation (like the alert correlation [CM02, NCRX04]), USAID detection, manually labelling.

- PRACTICAL VALIDATION OF THE INTEGRAL INTRUSION DETECTION SYSTEM. We will combine all proposed solutions into an integral intrusion detection system as in Figure 9.6. Our analysis and evaluation results should be considered in developing this system. At the same time, we will test the feasibility of collaboration between several distributed detection engines because the collaboration is useful to detect distributed attacks, DDoS, portscan for example.

10.4 Concluding Remarks

This thesis documented our research effort in analyzing and solving the generic problems in intrusion detection. A comprehensive framework is first introduced as the basis of our research. Then, based on the framework, the theoretical and experimental problem analysis is performed and verified. Using the analysis results, several practical solutions have also been proposed, designed and validated in our research. While the framework has shown great potential to build efficient intrusion detection solutions, there are still several open problems for future research.

References

- [ABE04] Nahla Ben Amor, Salem Benferhat, and Zied Elouedi. Naive bayes vs decision trees in intrusion detection systems. In *Proceedings of the 2004 ACM symposium on Applied computing, Special Track on Computer Security*, pages 420–424, Nicosia, Cyprus, March 14-17, 2004. ACM Press.
- [AC03] Muhammad H. Arshad and Philip K. Chan. Identifying outliers via clustering for anomaly detection. Technical Report CS-2003-19, Computer Science Department, Florida Institute of Technology, USA, 2003.
- [AJ00] Ramesh Agarwal and Mahesh V. Joshi. Pnrule: A new framework for learning classifier models in data mining. Technical Report 00-015, Department of Computer Science, University of Minnesota, USA, 2000.
- [And80] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P Anderson Co., Fort Washington, Pennsylvania, April 1980.
- [Axe99] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 1–7, Kent Ridge Digital Labs, Singapore, 1999. ACM Press.
- [Axe00a] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers University of Technology, Sweden, 2000.
- [Axe00b] Stefan Axelsson. A preliminary attempt to apply detection and estimation theory to intrusion detection. Technical report, Chalmers University of Technology, Göteborg, Sweden, 2000.
- [AZGL02] Kevin P. Anchor, Jesse B. Zydallis, Gregg H. Gunsch, and Gary B. Lamont. Extending the computer defense immune system: Network intrusion

REFERENCES

- detection with a multiobjective evolutionary programming approach. In *Proceedings of 1st International Conference on Artificial Immune Systems Conference Proceedings*, pages 12–21, University of Kent at Canterbury, Inglaterra, September 2002.
- [CC03] Suresh N. Chari and Pau-Chen Cheng. BlueBox: A Policy-Driven, Host-based Intrusion Detection System. *ACM Transaction on Infomation and System Security*, 6(2):173–200, May 2003.
- [CM02] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of 2002 IEEE Symposium on Security and Privacy*, pages 187 – 200, May 2002.
- [Coh95] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann.
- [DCF05] Hervé Debar, David A. Curry, and Benjamin S. Feinstein. The intrusion detection message exchange format. <http://www.ietf.org/ietf/lid-abstracts.txt>, January 27 2005.
- [DDW00] Hervé Debar, Marc Dacier, and Andreas Wespi. A revised taxonomy for intrusion detection systems. *Annales des Télécommunications*, 55(7-8):83–100, 2000.
- [Den87] Dorothy E. Denning. An intrusion detection model. *IEEE Transaction on Software Engineering*, SE-13(2):222–232, Feburary 1987.
- [Des03] Neil Desai. Intrusion prevention systems: the next step in the evolution of IDS. <http://www.securityfocus.com/infocus/1670>, Feb, 27 2003.
- [EAP⁺02] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Salvatore Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In D. Barbara and S. Jajodia (editors), *Applications of Data Mining in Computer Security*, Kluwer, 2002.
- [EFH02] Fernando Esponda, Stephanie Forrest, and Paul Helman. A formal framework for positive and negative detection schemes. *IEEE Transactions on Systems, Man and Cybernetics*, 34(1):357–373, Feburary 2002.

REFERENCES

- [Elk99] Charles Elkan. Results of the KDD'99 classifier learning contest. <http://www-cse.ucsd.edu/users/elkan/clresults.html>, September 1999.
- [FHSL96] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.
- [FKF⁺03] Henry Hanping Feng, Oleg M. Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly detection using call stack information. In *Proceedings of 2003 IEEE Symposium on Security and Privacy*, pages 62–77, Oakland, CA, 2003.
- [FPAC94] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonsel self discriminant in a computer. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, pages 202–212. Oakland, CA, 16-18 May 1994.
- [Gol89] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Pub. Co., 1989.
- [GSQ⁺04] Guofei Gu, Monirul I. Sharif, Xinzhou Qin, David Dagon, Wenke Lee, and George F. Riley. Worm detection, early warning and response based on local victim information. In *Proceedings of 20th Annual Computer Security and Applications Conference*, pages 136–145, Arizona, USA, December 2004.
- [HDL⁺90] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 296–304, May 1990.
- [HF00] Steven A. Hofmeyr and Stephanie Forrest. Architecture for an artificial immune system. *Evolutionary Computation*, 8(4):443–473, Winter, 2000.
- [HFS98] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

REFERENCES

- [HJS⁺93] Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josephine Ford. NADIR: An automated system for detecting network intrusions and misuse. *Computers and Security*, 12(3):253 – 248, May 1993.
- [HLF⁺01] Joshua W. Haines, Richard Lippmann, David J. Fried, Eushiuan Tran, Steve Boswell, and Marc A. Zissman. 1999 DARPA Intrusion Detection Evaluation: Design and Procedures. Technical Report 1062, Lincoln Laboratory, MIT, USA, Feb. 26 2001.
- [Jul03] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transaction on Information and System Security*, 6(4):443–471, November 2003.
- [JV93] Harold S. Javits and Alfonso Valdes. The NIDES statistical component: Description and justification. SRI Anual Report A010, SRI International, Computer Science Laboratory, March 1993.
- [KC04] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.
- [Ken99] Kris Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master thesis, Massachusetts Institute of Technology, June 1999.
- [KK04] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of 13th USENIX Security Symposium*, pages 271–286, San Diego, CA, USA, August 2004.
- [KMRV03] Christopher Kruegel, Darren Mutz, William Robertson, and Fredrik Valeur. Bayesian event classification for intrusion detection. In *Proceedings of 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2003.
- [KRL97] Calvin Ko, Manfred Ruschitzka, and Karl Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings of 1997 IEEE Symposium on Security and Privacy*, Oakland, CA, May 04-07 1997.

REFERENCES

- [KTK02] Christopher Kruegel, Thomas Toth, and Engin Kirda. Service specific anomaly detection for network intrusion detection. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 201–208, Madrid, Spain, March 2002.
- [LD04a] Zhuowei Li and Amitabha Das. Analyzing and improving the performance of a class of anomaly-based intrusion detectors. In *CoRR cs.CR/0410068*, October 2004.
- [LD04b] Zhuowei Li and Amitabha Das. Visualizing and identifying intrusion context from system calls trace. In *Proceedings of the 20th Annual Computer Security Applications Conference*, pages 61–70, Arizona, USA, December 2004.
- [LD05] Zhuowei Li and Amitabha Das. M of n features vs. intrusion detection. In *Proceedings of ICCSA in VRSAL 2005 Workshop*, Singapore, May 2005. LNCS, Springer-Verlag.
- [LDE05] Zhuowei Li, Amitabha Das, and Sabu Emmanuel. Model redundancy vs. intrusion detection. In *Proceedings of 1st Information Security Practice and Experience Conference - ISPEC*, Singapore, April 2005. Lecture Notes in Computer Science.
- [LDN03] Zhuowei Li, Amitabha Das, and Sukumar Nandi. Utilizing statistical characteristics of n-grams for intrusion detection. In *proceedings of the 2003 International Conference on CYBERWORLDS*, pages 486–493, Singapore, 2003. IEEE Computer Society.
- [LDZ05a] Zhuowei Li, Amitabha Das, and Jianying Zhou. Model generalization and its implications for intrusion detection. In *Proceedings of 3rd Applied Cryptography and Network Security (ACNS)*, pages 222–237, Columbia University, NY, USA, June 2005. Lecture Notes in Computer Science.
- [LDZ05b] Zhuowei Li, Amitabha Das, and Jianying Zhou. Theoretical basis for intrusion detection. In *Proceedings of 6th IEEE Information Assurance Workshop (IAW)*, West Point, NY, USA, June 2005. IEEE SMC Society.
- [LDZ05c] Zhuowei Li, Amitabha Das, and Jianying Zhou. USAID: Unifying signature-based and anomaly-based intrusion detection. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-05)*, pages 702–712, Hanoi, Vietnam, May 2005. LNAI/LNCS, Springer.

REFERENCES

- [Lib] Libpcap. <http://sourceforge.net/projects/libpcap/>.
- [LMS00] Wenke Lee, Matt Miller, and Salvatore J. Stolfo. Toward cost-sensitive modeling for intrusion detection. Technical Report No. CUCS-002-00, Computer Science, Columbia University, USA, 2000.
- [LS00] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4):227–261, November 2000.
- [LSM99] W. Lee, S.J. Stolfo, and K.W. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.
- [LV02] Yihua Liao and V. Rao Vemuri. Using text categorization techniques for intrusion detection. In *Proceedings of the 11th USENIX Security Symposium*, pages 51–59, August 2002.
- [LY01] Xiangyang Li and Nong Ye. Decision tree classifiers for computer intrusion detection. *Journal of Parallel and Distributed Computing Practices*, 4(2):179–190, 2001.
- [MC02a] Matthew V. Mahoney and Philip K. Chan. Learning models of network traffic for detecting novel attacks. Technical Report CS-2002-08, Computer Science Department, Florida Institute of Technology, USA, August 2002.
- [MC02b] Matthew V. Mahoney and Philip K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of ACM SIGKDD*, pages 376–385, July 23-26 2002.
- [McH01] John McHugh. Intrusion and intrusion detection. *International Journal of Information Security, Springer*, 1(1):14–35, 2001.
- [MHL94] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, May/June 1994.
- [Mit97] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [MS03] Srinivas Mukkamala and Andrew H. Sung. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of Digital Evidence*, 1(4):1–17, Winter, 2003.

REFERENCES

- [NCR02] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, Washington, DC, USA, November 2002.
- [NCRX04] Peng Ning, Yun Cui, Douglas S. Reeves, and Dingban Xu. Techniques and tools for analyzing intrusion alerts. *ACM Transactions on Information and System Security*, 7(2):274–318, May 2004.
- [NKS05] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 226–241, Okaland, CA, USA, May 2005.
- [NKS06] James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In *Proceedings of the 9th International Symposium On Recent Advances In Intrusion Detection (RAID 2006)*, September 2006.
- [NS05] James Newsome and Dawn Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2005.
- [NXHA04] Peng Ning, Dingban Xu, Christopher G. Healey, and Robert St. Amant. Building attack scenarios through integration of complementary alert correlation method. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2004.
- [Pax99] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks, Elsevier*, 31(23-24):2435–2463, 1999.
- [PDL⁺06] Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla, and Monirul I. Sharif. Misleadingworm signature generators using deliberate noise injection. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P)*, pages 17–31, 2006.
- [PN97] Phillip A. Porras and Peter G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, October 1997.

REFERENCES

- [Pow02] Richard Power. Computer security issues& trends. 2002 CSI/FBI computer crime and security survey, Computer Security Institute, 600 Harrison St. San Francisco CA 94107, 2002.
- [Rea] RealSecure. <http://www.realsecure.com>.
- [RJM04] Shai Rubin, Somesh Jha, and Barton P. Miller. Automatic generation and analysis of nids attacks. In *Proceedings of the 20th Annual Computer Security Applications Conference*, pages 28–38, Tucson, Arizona, USA, December 2004. IEEE Computer Society.
- [RJM05] Shai Rubin, Somesh Jha, and Barton P. Miller. Language-based generation and evaluation of nids signatures. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–17, Okaland, CA, USA, May 2005. IEEE Computer Society.
- [Roe99] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System Administration*, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.
- [RW05] Frank Ruskey and Mark Weston. Venn diagrams. *The Electronic Journal of Combinatorics, Dynamic Surveys*, DS5, June 2005.
- [SB01] Sasha and Beetle. A strict anomaly detection model for intrusion detection systems. http://www.totse.com/en/hack/hack_attack/162534.html, 2001.
- [SEVS04] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *Proceedings of the 6th Symposium on Operating System Design and Implementation*, pages 45–60, San Francisco, California, USA, December 2004. USENIX Association.
- [SGF⁺02] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 265–274, Washington, DC, USA, November 2002. ACM Press.
- [SHF97] Anil Somayaji, Steven Hofmeyr, and Stephanie Forrest. Principles of a computer immune system. In *Meeting on New Security Paradigms, 23-26 Sept. 1997, Langdale, UK*, pages p.75–82. ACM Press, 1997.

REFERENCES

- [SKAF⁺04] Tao Song, Calvin Ko, Jim Alves-Foss, Cui Zhang, and Karl N. Levitt. Formal reasoning about intrusion detection systems. In *RAID*, pages 278–295, 2004.
- [Sma88] Stephen E. Smaha. Haystack: An intrusion detection system. In *Proceedings of Fourth Aerospace Computer Security Applications Conference*, pages 37–44, Orlando, Florida, USA, December 1988.
- [SMPW04] Stuart Staniford, David Moore, Vern Paxson, and Nicholas Weaver. The top speed of flash worms. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malware*, pages 33–42, 2004.
- [SS04] Maheshkumar Sabhnani and Gürsel Serpen. Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set. *Journal of Intelligent Data Analysis*, 8(4):403–415, 2004.
- [SY05] Sufatrio and Roland H. C. Yap. Improving host-based ids with argument abstraction to prevent mimicry attacks. In *Proceedings of 8th International Symposium on Recent Advances in Intrusion Detection*, September 2005.
- [Szo05] Peter Szor. *The Art of Computer Virus Research and Defense*. Symantec Press. Addison Wesley Professional, Feb 2005.
- [TM03] Kymie M. C. Tan and Roy A. Maxion. Determining the operational limits of an anomaly-based intrusion detector. *IEEE Journal on selected areas in communications*, 21(1):96–110, January 2003.
- [TMK02] Kymie M. C. Tan, John Mchugh, and Kevin S. Killourhy. Hiding intrusions: From the abnormal to the normal and beyond. In *Proceedings of the 2002 Information Hiding*, pages 1–17, October 2002.
- [VK99] Giovanni Vigna and Richard A. Kemmerer. Netstat: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999.
- [VL89] H. S. Vaccaro and Gunar E. Liepins. Detection of anomalous computer session activity. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 280–289, Oakland, CA, USA, 1989.

REFERENCES

- [VRB04] Giovanni Vigna, William Robertson, and Davide Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 21–30, Washington DC, USA, 2004. ACM Press.
- [VS00] Alfonso Valdes and Keith Skinner. Adaptive, model-based monitoring for cyber attack detection. In *Proceeding of International Symposium on Recent Advances in Intrusion Detection*, pages 80–92, Toulouse, France, October 2000. Springer-Verlag.
- [WD01] David Wagner and Drew Dean. Intrusion detection via static analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 156–168, May 2001.
- [WFP99] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 133–145, May 1999.
- [Whi99] Whisker. A look at whisker’s anti-ids tactics. For details, please refer to <http://www.wiretrip.net/rfp/txt/whiskerids.html>, 1999.
- [Win] Winpcap. <http://www.winpcap.org/>.
- [WS02] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, Washington, DC, USA, November 2002.
- [WS04] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of International Symposium on Recent Advances in Intrusion Detection*, pages 203–222, Sophia Antipolis, France, September 2004.
- [XDC04] Haizhi Xu, Wenliang Du, and Steve J. Chapin. Context sensitive anomaly monitoring of process control flow to detect mimicry attacks and impossible paths. In *Proceedings of International Symposium on Recent Advances in Intrusion Detection*, pages 21–38, Sophia Antipolis, France, September 2004.

Appendix A

A.1 The Sorted Feature Vector

From all features in Table 3.3, we have selected the following features in varUSAID in our experiments according to their distinguishing capability.

Index in FV_{sort}	Index in FV	Feature
1	4	Source Bytes
2	2	Service
3	3	Flag
4	26	REJ Error Rate
5	40	Destination Host Service REJ Error Rate
6	28	Same Service Rate
7	36	Destination Host Service Different Host Rate
8	29	Different Service Rate
9	30	Service Different Host Rate
10	5	Destination Bytes
11	1	Protocol Type
12	25	Service SYN Error Rate
13	38	Destination Host Service SYN Error Rate
14	11	Logged In
15	6	Land
16	24	SYN Error Rate
17	0	Duration

Table A.1: The features in the sorted feature vector FV_{sort} .

A.2 Acronyms

In this section, we list the acronyms used in this thesis.

CHAPTER A.

Acronym	Meaning
<i>New items introduced in Chapter 1.</i>	
IDS	Intrusion Detection System
SID	Signature-based Intrusion Detection
AID	Anomaly-based Intrusion Detection
USAID	Unified Signature-based and Anomaly-based Intrusion Detection
varUSAID	Variable-length Signatures for USAID.
NSM	Network Security Monitor
EMERALD	Event Monitoring Enabling Responses to Anomalous Live Disturbances
MADAM ID	Mining Audit Data for Automated Models for Intrusion Detection
LERAD	LEarning Rules for Anomaly Detection
BSM	Basic Security Module
<i>New items introduced in Chapter 2.</i>	
IDES	Intrusion Detection Expert System
NIDES	Next-Generation Intrusion Detection Expert System
NADIR	Network Anomaly Detection and Intrusion Reporter
PAYL	Payload-based Intrusion Detection
ACL2	Applicative Common Lisp 2
FSA	Finite State Automaton
VtPath	Virtual Path
RIPPER	A system for fast effective rule induction
STAT	State Transition Analysis Technique
FST	Finite State Transducer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
<i>New items introduced in Chapter 3.</i>	
FV	Feature Vector, which includes a set of features
NSA	Normal, Suspicious and Attack
ICMP	Internet Control Message Protocol
NETBIOS	Network Basic Input Output System
DHCP	Dynamic Host Configuration Protocol
DOS	Denial of Service
U2R	User to Root
R2L	Remote to Local
<i>New items introduced in Chapter 4.</i>	
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
DR	Detection Rate
FAR	False Alarm Rate
<i>New items introduced in Chapter 9.</i>	
ACI	Anomaly Context Identification
SSO	Security Site Officer
GUI	Graphical User Interface
IDMEF	Intrusion Detection Message Exchange Format
FRGN	Foreign Sequences
SELF	Self Sequences
MFS	Minimum Foreign Sequence
FSG	Foreign Sequence Graph
CERT	Computer Emergency Response Team
MTTA	Mean Time To Alarm
DDoS	Distributed Denial of Service

Table A.2: A list of acronyms.

Appendix B

List of Publications

Papers in Refereed Journals

- (1) Zhuowei Li, Amitabha Das, *Anomaly Context Identification for Suppressing False Alarms in Intrusion Detection*, In submission.
- (2) Zhuowei Li, Amitabha Das, and Jianying Zhou, *USAID: A Comprehensive Framework for Intrusion Detection and Its Applications*, major revision.
- (3) Zhuowei Li, Amitabha Das, *Analyzing and Evaluating Dynamics in stide Performance for Intrusion Detection*, Knowledge-based Systems, Elsevier, Vol. 19, No. 7, pp. 576-591, 2006.
- (4) Zhuowei Li, Amitabha Das, *The Utility of Partial Knowledge in Behavior Models: An Evaluation for Intrusion Detection*, International Journal of Network Security, Vol. 1, No. 3, pp. 138-146, 2005.

Papers in Refereed Conferences

- (5) Zhuowei Li, Amitabha Das, and Jianying Zhou, *Towards Automatic Assembly of Privacy-Preserved Signatures.*, Accepted by 4th International Conference on Trust, Privacy and Digital Business, 3-7 September, 2007.
- (6) Zhuowei Li, Amitabha Das, and Jianying Zhou, *Evaluating Model Generalization on Intrusion Detection Performance*, Accepted by 22nd IFIP TC-11 International Information Security Conference, 14-16 May, 2007.
- (7) Zhuowei Li, Amitabha Das, and Jianying Zhou, *Theoretical Basis for Intrusion Detection*, In Proceedings of 6th IEEE Information Assurance Workshop (IAW). June 15-17, 2005. West Point, NY.

CHAPTER B. LIST OF PUBLICATIONS

- (8) Zhuowei Li, Amitabha Das, and Jianying Zhou, *Model Generalization and its Implications for Intrusion Detection*, In Proceedings of Applied Cryptography and Network Security (ACNS), Pages 222-237, June 7-10, 2005.
- (9) Zhuowei Li, Amitabha Das, and Jianying Zhou, *Unifying Signature-based and Anomaly-based Intrusion Detection*, In Proceedings of 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Pages 702-712, Hanoi, Vietnam, May 2005.
- (10) Zhuowei Li, Amitabha Das, *M of N Features vs. Intrusion Detection*, In Proceedings of International Conference on Computational Science and its Applications (ICCSA), Pages 994-1003, May 2005.
- (11) Zhuowei Li, Amitabha Das, *Model Redundancy vs. Intrusion Detection*, In Proceedings of Information Security Practice and Experience Conference (ISPEC), Pages 217-229, April 2005.
- (12) Zhuowei Li, Amitabha Das, *Visualizing and Identifying Intrusion Context from System Calls Trace*, In Proceedings of 20th Annual Computer Security Applications Conference (ACSAC), Pages 61-70, Arizona. December 6-10, 2004.
- (13) Zhuowei Li, Amitabha Das, and Sukumar Nandi, *Utilizing Statistical Characteristics of N-grams for Intrusion Detection*, In proceedings of 2003 International Conference on CYBERWORLDS, Pages:486-493,December 3-5, 2003.

Others

- (14) Zhuowei Li, *Evolving Behavior Models for Intrusion Detection*, Presented in WiP session of 20th Annual Computer Security Applications Conference, Tucson, Arizona. December 6-10, 2004.