

Using Artificial Anomalies to Detect Unknown and Known Network Intrusions

W. Fan¹, M. Miller², S. Stolfo², W. Lee³, and P. Chan⁴

¹ IBM T.J.Watson Research, Hawthorne, NY 10532, weifan@us.ibm.com

² Computer Science, Columbia University, New York, NY, {mmiller,sal}@cs.columbia.edu

³ College of Computer Science, Georgia Tech, Atlanta, GA, wenke@cc.gatech.edu

⁴ Computer Science, Florida Institute of Technology, Melbourn, FL, pkc@cs.fit.edu

Abstract. Intrusion detection systems (IDSs) must be capable of detecting new and unknown attacks, or anomalies. We study the problem of building detection models for both pure anomaly detection and combined misuse and anomaly detection (i.e., detection of both known and unknown intrusions). We show the necessity of artificial anomalies by discussing the failure to use conventional inductive learning methods to detect anomalies. We propose an algorithm to generate artificial anomalies to coerce the inductive learner into discovering an accurate boundary between known classes (normal connections and known intrusions) and anomalies. Empirical studies show that our pure anomaly detection model trained using normal and artificial anomalies is capable of detecting more than 77% of all unknown intrusion classes with more than 50% accuracy per intrusion class. The combined misuse and anomaly detection models are as accurate as a pure misuse detection model in detecting known intrusions and are capable of detecting at least 50% of unknown intrusion classes with accuracy measurements between 75% and 100% per class.

Keywords: anomaly detection, intrusion detection, artificial anomaly, security

1. Introduction

Data analysis tasks can be broadly categorized into anomaly detection and classification problems. *Anomaly detection* tracks events that are inconsistent with or deviate from events that are known or expected. For example, in intrusion

Received xxx

Revised xxx

Accepted xxx

detection anomaly detection systems flag observed activities that deviate significantly from established normal usage profiles. On the other hand, classification systems use patterns of well-known classes to match and identify known labels for unlabeled datasets. In intrusion detection, classification of known attacks is also called *misuse detection*.

Anomaly detection systems are not as well studied, explored, or applied as classification systems. Most of the leading commercial intrusion detection systems (IDSs) employ solely misuse detection techniques, which use patterns of "known" attacks to detect intrusions. However, as anecdotes of serious break-ins to major government, military, and commercial sites have shown, our adversaries, knowing that intrusion prevention and detection systems are installed in our networks, will always be attempting to develop and launch "new" attacks. Last year's Distributed Denial-of-Service (DDOS) attacks have caused major disruptions for services provided over the Internet.

In the generation of classification models, training data containing instances of known classes is often available for training (or human analysis) and the goal is simply to detect instances of these known classes. Anomaly detection, however, relies on data belonging to one single class (such as purely *normal* connection records) or limited instances of some known classes with the goal of detecting all unknown classes. It is difficult to use traditional inductive learning algorithms for such a task, as most are only good at distinguishing the boundaries among all given classes of data. In this paper, we explore the use of traditional inductive learning algorithms for anomaly detection by working from the dataset level. We present methods for generating artificial anomalies based on known classes to coerce an arbitrary machine learning algorithm to learn hypotheses that separate all known classes from unknown classes. We discuss the generation of anomaly detection models from pure normal data, and also discuss the generation of combined misuse and anomaly detection models from data that contains known classes. We apply the proposed approaches to network-based intrusions.

The rest of the paper is organized as follows. Section 2 discusses the motivation for artificial anomaly generation and the different methods. In Sections 3-6, we evaluate our methods using RIPPER (Cohen, 1995), an inductive rule learner, trained and tested with the 1998 DARPA Intrusion Detection Evaluation dataset. Section 7 reviews related work in anomaly detection. Section 8 offers conclusive remarks and discusses avenues for future work.

2. Artificial Anomaly Generation

A major difficulty in using machine learning methods for anomaly detection lies in making the learner discover boundaries between known and unknown classes. Since we begin without any examples of anomalies in our training data (by definition of the task of anomaly detection), a machine learning algorithm will only uncover boundaries that separate different known classes in the training data. This behavior is intended to prevent overfitting a model to the training data. Learners only generate hypotheses for the provided class labels in the training data. These hypotheses define decision boundaries that separate the given class labels. To achieve generalization and avoid overfitting, learning algorithms usually do not specify a boundary beyond that necessary to separate known classes.

Some learners can generate a default classification for instances that are not covered by the learned hypothesis. The label of this default classification is often

defined to be the most frequently occurring class of all uncovered instances in the training data. It is possible to modify this default prediction to be *anomaly*, signifying that any uncovered instance should be considered anomalous. It is also possible to tune the parameters of some learners to coerce them into learning more specific hypotheses. As shown in Section 6.1, our experimentation with these methods does not yield a reasonable performance.

The failure of using more specific hypotheses and modifying a model’s default prediction has motivated us to propose *artificial anomaly generation* for such a task. Artificial anomalies are injected into the training data to help the learner discover a boundary around the original data. All artificial anomalies are given the class label *anomaly*. Our approach to generating artificial anomalies focuses on “near misses,” instances that are close to the known data, but are not in the training data. We assume the training data are representative; hence near misses can be safely assumed to be anomalous. Our artificial anomaly generation methods are independent of the learning algorithm, as the anomalies are merely added to the training data.

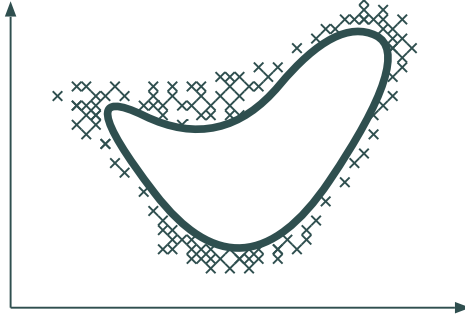
2.1. Naive Artificial Anomaly Generation

One simple method for generating artificial anomalies is to produce random instances that span the complete domain defined by all features. A value for each feature is chosen randomly from the definition domain of that feature. The artificial anomalies produced will be uniformly distributed across the domain defined by all features. If instances of known classes are clustered into small areas of that domain, and a sufficient amount of artificial anomalies is produced, the learner should be able to uncover the boundary between known and unknown classes as it tries to separate known classes from artificial anomalies. As discussed in Section 6.3, this method is somewhat effective. The domain of the decision space, the complexity of the target decision boundary, the amount of known data, and the amount of artificial anomalies produced all play important roles in the effectiveness of this approach. Since we do not know the decision boundary or the true domain of the space in which all known classes reside, it is difficult to estimate how many artificial anomalies are sufficient for effective learning.

2.2. Distribution-based Artificial Anomaly Generation

An alternative approach is to generate artificial anomalies that are based on the known data. Since we do not know where the exact decision boundary is between the known and anomalous instances, we assume that the boundary may be very close to the existing data. To generate artificial anomalies close to the known data, a useful heuristic is to randomly change the value of one feature of an example while leaving the other features unaltered.

Some regions of known data in the instance space may be sparsely populated. We compare sparse regions to small islands, and dense regions to large islands, in an ocean. To avoid overfitting, learning algorithms are usually biased towards discovering more general hypotheses. Since we only have known data, we want to prevent hypotheses from being *overly* general when predicting these known classes. That is, sparse regions may be grouped into dense regions to produce singularly large regions covered by overly general hypotheses. Using our analogy,



Curved line is the border of instance space.
 The area enclosed in the curved line is the instance space.
 × are artificial anomalies (more at sparse regions).

Fig. 1. Artificial Anomalies

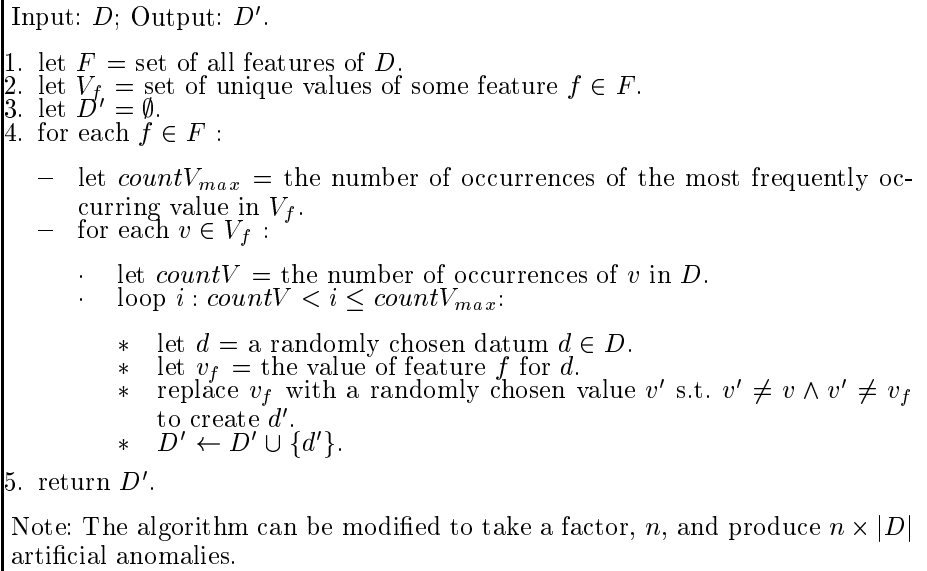


Fig. 2. Distribution-based Artificial Anomaly Algorithm (DBA2)

small islands are unnecessarily grouped into large islands to form apparently larger islands. It is possible to produce artificial anomalies around the edges of these sparse regions and coerce the learning algorithm to discover the specific boundaries that distinguish these regions from the rest of the instance space. In other words, we want to generate data that will amplify these sparse regions. The intuitive idea of sparse regions and artificially anomalies generated around sparse regions is illustrated in Figure 1.

Sparse regions are characterized by infrequent values of individual features. To amplify sparse regions, we proportionally generate more artificial anomalies

around sparse regions depending on their sparsity using a proposed algorithm presented in detail in Figure 2. Assuming that the value v of some feature f is infrequently present in the dataset, we calculate the difference between the number of occurrences of v , $countV$, and the number of occurrences of the most frequently occurring value v_{max} of the given feature, $countV_{max}$. We then randomly sample $countV_{max} - countV$ data points from the training set. For each data point d in this sample, we replace the value of feature f , v_f , with any v' such that $v' \neq v \wedge v' \neq v_f$ to generate an artificial anomaly, d' . The learning algorithm used will then specifically cover all instances of the data with value v for feature f . This anomaly generation process is called *distribution-based* artificial anomaly generation, as the distribution of a feature’s values across the training data is used to selectively generate artificial anomalies.

One way to verify the need to generate $countV_{max} - countV$ anomalies for each value v is to produce a uniform number of anomalies for each v . Our experiments show that this method is ineffective. Our surmise is that artificial anomalies computed uniformly will not effectively force the learning algorithm to distinguish sparse regions from the rest of the example space.

2.3. Desiderata

One critical assumption of the artificial anomaly generation algorithm is that the original training data is sufficient and representative. When this assumption is violated, artificial anomalies will intersect with the original training data, and the computed anomaly detection models are very likely to incur higher false alarm rate. Data sufficiency is a general problem for many machine learning algorithms including distribution-based artificial anomaly generation algorithm. One simple method to test data sufficiency is to plot the histogram of each feature’s values. When there is no significant change in the histogram, the data can be assumed sufficient.

The total number of artificial anomalies depends on the number of unique feature values in the training data. Assuming that feature f has $|V_f|$ unique feature values. Then the maximum number of artificial anomalies computed for feature f is $|V_f| \cdot countV_{max} - \sum_{v \in V_f} countV(v)$, which is at most $|V_f| \cdot countV_{max}(f)$. Taking into account every feature, the maximum number of anomalies is therefore $\sum_{f \in F} |V_f| \cdot countV_{max}(f)$. In practice, we don’t need all artificial anomalies. The inner loop of the DBA2 algorithm (loop $i : countV \leq i \leq countV_{max}$) can take a parameter to generate only a portion of the artificial anomalies. Our experiments show that the amount of artificial anomalies do not significantly influence the accuracy of the learned models.

2.4. Filtered Artificial Anomalies

In the above discussion, we assume that artificial anomalies do not intersect with any known data. In the DARPA dataset (See Section 3), there are 41 features. A few of these features have two possible values, but most of them are either continuous or contain more than 5 values. Since the total size of the training data is significantly smaller than 2^{41} (the lower bound of the feature domain), the chance of intersection is very small. We could always check for collision with known instances, but this is a very expensive process. Another approach is to

Table 1. Intrusions, Categories and Sampling

U2R		R2L		DOS		PRB	
buffer_overflow	1	ftp_write	4	back	1	ipsweep	1
loadmodule	2	guess_passwd	1	land	1	nmap	1
multihop	6	imap	2	neptune	$\frac{1}{20}$	portsweep	1
perl	6	phf	3	pod	1	sat an	1
rootkit	2	spy	8	smurf	$\frac{1}{20}$		
		warezclient	1	teardrop	1		
		warezmaster	1				

filter artificial anomalies with hypotheses learned on the original data. We use the training set plus an initial generation of artificial anomalies to learn a model. We then evaluate this model over previously generated artificial anomalies and remove any anomalies classified as some known class. This process is repeated until the size of the the set of artificial anomalies remains relatively stable.

3. Experimental Setup

For generation of our models, we have chosen to use RIPPER (Cohen, 1995), an inductive decision tree learner. RIPPER can learn both *unordered* and *ordered* rulesets. The use of these types of rulesets in ID has been discussed in our previous work (Fan, Lee, Stolfo and Miller, 2000). In all reported results, unless clearly stated, we always use an *unordered* RIPPER ruleset and inject an amount of distribution-based artificial anomalies equal to the size of the training set (i.e., we use DBA2 with $n = 1$).

Our experiments use data distributed by the 1998 DARPA Intrusion Detection Evaluation Program, which was conducted by MIT Lincoln Lab (available from the UCI KDD repository as the 1999 KDD Cup Dataset). We use the same taxonomy for categorization of intrusions as was used by the DARPA evaluation. This taxonomy places intrusions into one of four categories: denial of service (DOS), probing (PRB), remotely gaining illegal remote access to a local account or service (R2L), and local user gaining illegal root access (U2R). The DARPA data were gathered from a simulated military network and includes a wide variety of intrusions injected into the network over a period of 7 weeks. The data was then processed into connection records using MADAM ID (Lee, 1999). A 10% sample was taken which maintained the same distribution of intrusions and normal connections as the original data (this sample is available as kdd-cup.data.10% from the UCI KDD repository). We used 80% of this sample as training data and left the remaining 20% unaltered to be used as test data for evaluation of learned models. For infrequent intrusions in the training data, the records for those connections were repeatedly injected to prevent the learning algorithm from neglecting them as statistically insignificant and not generating any rules for them. For overwhelming intrusions in the training data, only 1 out of 20 records were sampled. This is an ad hoc approach, but it produces reasonable results. Table 1 shows the category (U2R, R2L, DOS, PRB) and sampling rate of each intrusion.

Table 2. Anomaly Detection Rate and False Alarm Rate of Pure Anomaly Detection

		$\%a_{ttl}$	94.26			$\%far$	2.02
(a)							
		$\%a$	$\%a$	$\%a$	$\%a$	$\%a$	$\%a$
buffer_overflow	100.00√	ftp_write	50√	back	100.00√	ipsweep	-
loadmodule	66.67√	guess_passwd	100.00√	land	75.00√	nmap	-
multihop	57.14√	imap	83.33√	neptune	80.52√	portsweep	4.81
perl	-	phf	100.00√	pod	9.62	satan	0.32
rootkit	10.00	spy	-	smurf	99.94√		
		warezclient	64.25√	teardrop	-		
		warezmaster	80.00√				
U2R	47.06√	R2L	66.67√	DOS	94.31√	PRB	1.34

√: significant or $\%a \geq 50\%$
(b)

4. Pure Anomaly Detection

For pure anomaly detection, we learned a model using all available *normal* connections augmented by DBA2 anomalies generated from these normal connections. We refer to this collection as dataset₀. RIPPER learns a large number of rules for both *normal* and *anomaly* from this dataset.

4.1. Results

Table 2 shows the results of the pure anomaly detection model. We use detection rate and false alarm rate to evaluate performance. These terminologies are commonly adopted in the intrusion detection community. Anomaly detection rate, or percentage of occurrences of some unknown intrusion i that are detected as anomalies, is defined as $\%a_i = \frac{|A \cap W_i|}{|W_i|} \times 100\%$, where A is the set of all predicted anomalies and W_i is the set of occurrences of label i in the dataset. We omit the subscript i where the meaning is clear from the context. Similarly, we calculate cumulative anomaly detection rate over all unknown intrusions ($\%a_{ttl}$) and cumulative anomaly detection rate over different categories of unknown intrusions (such as $\%a_{u2r}$). Also, we measure the false alarm rate ($\%far$) of anomalous classifications. This is the percentage of predicted anomalies that are *normal* connections, and is defined as $\%far = \frac{|A \cap W_{normal}|}{|A|} \times 100\%$. If a measurement has a value of 0, we represent it with “-” to enhance readability of the presented tables.

The cumulative anomaly detection rate over all intrusions and false alarm rate are shown in Table 2(a). The anomaly detection model successfully detects 94.26% of all anomalous connections in the test data and has a false alarm rate of 2.02%. To examine the performance for specific intrusion classes and categories, the anomaly detection rate ($\%a$) for each class and category is shown in Table 2(b). The anomaly detection model is capable of detecting most intrusion classes, even though there are no intrusions at all in the training set. A total of 17 out of 22 intrusion classes (all non-null measurements) are detected as anomalies. For 13 out of 22 intrusions (all entries highlighted by √), the proposed method catches at least 50% of all occurrences. There are 3 intrusions (*guess_passwd*,

Table 3. Intrusion Clusters

1	2	3	4	5	6	7
back	buffer_overflow loadmodule perl rootkit	ftp_write warezclient warezmaster	guess_passwd	imap	land	portsweep satan
8	9	10	11	12	13	
ipsweep nmap	multihop	neptune	phf	pod teardrop	spy smurf	

buffer_overflow and *phf*) that our approach is capable of detecting perfectly (i.e., $\%a = 100\%$). These 3 intrusions belong to the more harmful U2R and R2L categories. The anomaly detection rates of all 4 categories of intrusions indicate that, in general, each category is successfully detected. In 3 out of 4 categories (U2R, R2L and DOS), the model detects more than or nearly 50% of all intrusion occurrences of that category. It is important to note that these three categories are the most damaging and most important types of intrusions to be detected.

5. Combined Misuse and Anomaly Detection

Pure anomaly detection might still have high false alarm rate; the boundaries implied by artificial anomalies can be sharpened by real intrusions. Separate modules for anomaly and misuse detection will not be as efficient as one single module that detects misuse and anomaly in the same time. All these have motivated us to explore the use of artificial anomalies for such as task.

We learn a single ruleset for combined misuse and anomaly detection. The ruleset has rules to classify a connection to be *normal*, one of the known intrusion classes, or *anomaly*. In order to evaluate this combined approach, we group intrusions together into a number of small *clusters* as shown in Table 3. We create datasets (dataset_{*i*}, $1 \leq i \leq 12$)¹ by incrementally adding each cluster_{*i*} into the *normal* dataset and re-generating artificial anomalies. This is to simulate the process of the invention of new intrusions and their incorporation into the training set. We later evaluate the effect of the ordering of the clusters on the experimental results. We learn models that contain misuse rules for the intrusions that are “known” in the training data, anomaly detection rules for unknown intrusions in left-out clusters, and rules that characterize normal behavior.

Each cluster contains intrusions that require similar features for effective detection. Clusters should not be confused with the attack categories in our taxonomy. One example of a cluster contains the following intrusions: *buffer_overflow*, *loadmodule*, *perl* and *rootkit*. These intrusions all attempt to gain unauthorized root access to a local machine and require features such as *root_shell* (whether a root shell is obtained) and *su_flag* (an indication of whether the *su root* command been used, in any of its derivations). Some clusters have completely disjoint feature sets, yet some intersect slightly. A model that is trained to detect intrusions from one cluster may have difficulties detecting intrusions from another cluster. For clusters with intersecting feature sets, we hope that a model learned

¹ We leave out the 13th cluster for testing.

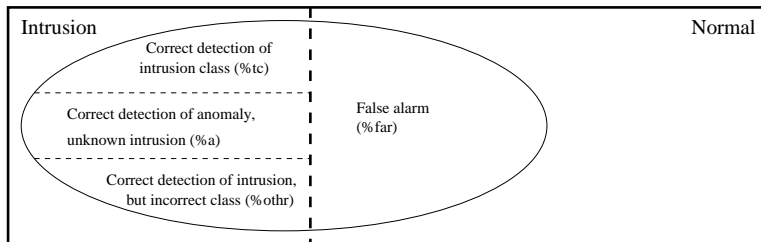


Fig. 3. Relationship of Metrics

using training instances of intrusions from some cluster may be used to detect intrusions of other clusters as anomalies.

Before explaining our results, we must define some frequently used terms. Any intrusion class that appears in the training data is a *known intrusion*. Similarly, any intrusion class not in the training set is an *unknown intrusion*, or *true anomaly*. *Predicted anomalies* include *true anomalies* and may also include instances of *known intrusions* and *normal*. We use *anomaly* to refer to *predicted anomaly* where our intention is clear from context.

5.1. Results

Our results for combined misuse and anomaly detection methods are shown in Tables 4-8 and Figures 4-6. Based on the outcome of detection, we calculate the following measurements: true class detection rate ($\%tc$), anomaly detection rate ($\%a$), and other class detection rate ($\%othr$). The relationship of these metrics is shown in the Venn diagram in Figure 3. The outside rectangle represents the set of data to be evaluated and the inside ellipse depicts the set of alarms generated by our learned detection models. True class detection rate measures the percentage of connection class i (*normal* or intrusions) being correctly predicted as its true class, and is defined as $\%tc = \frac{|P_i \cap W_i|}{|W_i|} \times 100\%$, where P_i is the set of predictions with label i . We use *na* (or *not available*) if the value of $\%tc$ is undefined, which is the case when an intrusion was not included in the training data. $\%a$ is defined as in Section 4.1, but it can be measured for both known and unknown intrusions or intrusion categories that are predicted as anomalies. Other class detection rate, or the rate of detection as another class of intrusion, is the percentage of occurrences of intrusion i that are detected as some class of intrusion other than its true label or anomaly, and is defined as $\%othr = \frac{\sum_{i' \neq i} |P_{i'} \cap W_i|}{|W_i|} \times 100\%$. Additionally, total detection rate is defined as $\%ttl = \%tc + \%a + \%othr$.

We examine our results from several perspectives. First, it is important to see how the proposed method influences the true class detection rate ($\%tc$) of known intrusions. Ideally, using artificial anomalies should allow anomaly detection to classify true anomalies without degrading the performance of detecting known intrusions with misuse rules. Second, we evaluate the effectiveness of detecting true anomalies. Third, we examine whether anomalous classification can compensate for low detection rates of known intrusions by misuse rules. Finally, we show the false alarm rates of anomaly detection in different test settings.

Table 4. Combined Misuse & Anomaly Detection Data

dataset ₁				dataset ₂				
	%tc	%a	%othr	%ttl	%tc	%a	%othr	%ttl
normal	99.04	0.96	na	na	99.25	0.74	na	na
back	98.64	1.36○	-	100.00	98.64	1.36○	-	100.00
buffer_overflow	na	100.00√	-	100.00	80.00	20.00○	-	100.00
loadmodule	na	77.78√	-	77.78	77.78	22.22○	-	100.00
perl	na	-	-	-	100.00	-	-	100.00
rootkit	na	10.00	-	10.00	70.00	-	-	70.00
ftp_write	na	62.50√	-	62.50	na	37.50	-	37.50
warezclient	na	69.08√	0.97	70.05	na	32.37	-	32.37
warezmaster	na	80.00√	-	80.00	na	100.00√	-	100.00
guess_passwd	na	100.00√	-	100.00	na	90.91√	-	90.91
imap	na	83.33√	-	83.33	na	100.00√	-	100.00
land	na	-	-	-	na	50.00√	-	50.00
portsweep	na	69.23√	-	69.23	na	69.23√	-	69.23
satan	na	12.97	-	12.97	na	17.09	-	17.09
ipsweep	na	20.16	-	20.16	na	46.37√	-	46.37
nmap	na	73.91√	-	73.91	na	80.43√	-	80.43
multihop	na	71.43√	-	71.43	na	71.43√	28.57	100.00
neptune	na	94.50√	-	94.50	na	98.39√	-	98.39
phf	na	100.00√	-	100.00	na	100.00√	-	100.00
pod	na	94.23√	-	94.23	na	100.00√	-	100.00
teardrop	na	61.93√	-	61.93	na	90.86√	-	90.86
spy	na	-	-	-	na	50.00√	-	50.00
smurf	na	81.21√	-	81.21	na	100.00√	-	100.00
U2R	na	52.94√	-	52.94	61.76	23.53	5.88	91.18
R2L	na	71.08√	0.80	71.89	na	40.96√	-	40.96
DOS	0.56	84.35√	-	84.91	0.56	98.98√	-	99.53
PRB	na	32.89	-	32.89	na	42.79√	-	42.79
%far		0.28				0.18		

dataset ₃				dataset ₄				
	%tc	%a	%othr	%ttl	%tc	%a	%othr	%ttl
normal	99.18	0.80	na	na	98.94	1.04	na	na
back	98.41	1.59○	-	100.00	98.87	1.13○	-	100.00
buffer_overflow	60.00	40.00○	-	100.00	60.00	40.00○	-	100.00
loadmodule	88.89	11.11○	-	100.00	77.78	22.22○	-	100.00
perl	100.00	-	-	100.00	100.00√	-	-	100.00
rootkit	80.00	-	-	80.00	60.00	-	10.00	70.00
ftp_write	100.00	-	-	100.00	100.00	-	-	100.00
warezclient	94.69	1.45	-	96.14	94.69	2.90	-	97.58
warezmaster	60.00	-	20.00	80.00	60.00	20.00○	20.00	100.00
guess_passwd	na	90.91√	-	90.91	90.91	9.09○	-	100.00
imap	na	100.00√	-	100.00	na	91.67√	-	91.67
land	na	25.00	-	25.00	na	75.00√	-	75.00
portsweep	na	69.23√	-	69.23	na	-	-	-
satan	na	11.71	-	11.71	na	-	-	-
ipsweep	na	53.63√	-	53.63	na	61.29√	-	61.29
nmap	na	93.48√	-	93.48	na	45.65	-	45.65
multihop	na	28.57	71.43	100.00	na	57.14√	42.86	100.00
neptune	na	94.48√	-	94.48	na	0.27	-	0.27
phf	na	100.00√	-	100.00	na	50.00√	-	50.00
pod	na	100.00√	-	100.00	na	94.23√	-	94.23
teardrop	na	-	-	-	na	35.53	-	35.53
spy	na	50.00√	-	50.00	na	50.00√	-	50.00
smurf	na	100.00√	-	100.00	na	100.00√	-	100.00
U2R	64.71	14.71	14.71	94.12	55.88	23.53	11.76	91.18
R2L	83.13	12.05	0.40	95.58	87.15	8.84	0.40	96.39
DOS	0.55	97.68√	-	98.23	0.56	71.97√	-	72.52
PRB	na	43.64√	-	43.64	na	21.15	-	21.15
%far		0.20				0.36		

dataset_i: normal + cluster_{1-i} + DBA2-generated anomalies from (normal + cluster_{1-i})

√: significant anomaly detection, or %a > 100%

○: significant compensation for misuse detection, or (%tc + %a ≈ 100% ∧ %tc < 100%) ∨ (%a ≥ 0.25 × %tc)

Table 5. Combined Intrusion & Anomaly Detection Data (cont.)

dataset ₅					dataset ₆			
	%tc	%a	%othr	%ttl	%tc	%a	%othr	%ttl
normal	98.86	1.13	na	na	98.84	1.15	na	na
back	98.64	1.36○	-	100.00	98.87	1.13○	-	100.00
buffer_overflow	60.00	20.00○	20.00	100.00	100.00	-	-	100.00
loadmodule	100.00	-	-	100.00	55.56	33.33○	11.11	100.00
perl	100.00	-	-	100.00	100.00	-	-	100.00
rootkit	90.00	-	-	90.00	20.00	60.00○	-	80.00
ftp_write	100.00	-	-	100.00	100.00	-	-	100.00
warezclient	96.14	2.90○	-	99.03	95.17	3.38	-	98.55
warezmaster	60.00	20.00○	20.00	100.00	60.00	-	20.00	80.00
guess_passwd	90.91	-	9.09	100.00	90.91	9.09○	-	100.00
imap	100.00	-	-	100.00	91.67	8.33○	-	100.00
land	na	-	-	-	50.00	-	-	50.00
portsweep	na	72.60√	-	72.60	na	94.71√	-	94.71
satan	na	89.56√	-	89.56	na	24.37	-	24.37
ipsweep	na	60.48√	-	60.48	na	64.11√	-	64.11
nmap	na	80.43√	-	80.43	na	91.30√	-	91.30
multihop	na	57.14√	14.29	71.43	na	42.86	57.14	100.00
neptune	na	5.21	0.01	5.22	na	79.36√	-	79.36
phf	na	100.00√	-	100.00	na	25.00	-	25.00
pod	na	69.23√	-	69.23	na	100.00√	-	100.00
teardrop	na	3.05	-	3.05	na	6.60	-	6.60
spy	na	100.00√	-	100.00	na	-	-	-
smurf	na	99.98√	-	99.98	na	100.00√	-	100.00
U2R	70.59	14.71	5.88	91.18	44.12	35.29	14.71	94.12
R2L	93.17	5.22	0.80	99.20	91.97	4.02	0.40	96.39
DOS	0.56	73.20√	-	73.76	0.56	93.55√	-	94.11
PRB	na	75.92√	-	75.92	na	58.07√	-	58.07
%far			0.38				0.30	

dataset ₇					dataset ₈			
	%tc	%a	%othr	%ttl	%tc	%a	%othr	%ttl
normal	99.29	0.70	na	na	99.19	0.79	na	na
back	98.64	1.36○	-	100.00	98.64	1.36○	-	100.00
buffer_overflow	80.00	20.00○	-	100.00	20.00	80.00○	-	100.00
loadmodule	88.89	11.11○	-	100.00	44.44	44.44○	11.11	100.00
perl	100.00	-	-	100.00	100.00	-	-	100.00
rootkit	na	80.00√	-	80.00	60.00	-	-	60.00
ftp_write	75.00	12.50	-	87.50	100.00	-	-	100.00
warezclient	96.62	1.45	-	98.07	96.14	2.42	-	98.55
warezmaster	60.00	-	20.00	80.00	60.00	20.00○	-	80.00
guess_passwd	90.91	9.09○	-	100.00	90.91	9.09○	-	100.00
imap	100.00	-	-	100.00	100.00	-	-	100.00
land	75.00	-	-	75.00	75.00	25.00○	-	100.00
portsweep	99.04	0.96○	-	100.00	99.04	0.48○	-	99.52
satan	99.37	0.32○	-	99.68	98.73	1.27○	-	100.00
ipsweep	na	56.05√	-	56.05	99.19	0.40○	-	99.60
nmap	na	52.17√	41.30	93.48	89.13	10.87○	-	100.00
multihop	na	57.14√	42.86	100.00	na	42.86√	28.57	71.43
neptune	na	19.17	0.20	19.37	na	98.85√	0.94	99.79
phf	na	100.00√	-	100.00	na	25.00	-	25.00
pod	na	100.00√	-	100.00	na	100.00√	-	100.00
teardrop	na	13.71	-	13.71	na	88.32√	6.60	94.92
spy	na	50.00√	-	50.00	na	50.00√	-	50.00
smurf	na	100.00√	-	100.00	na	100.00√	-	100.00
U2R	44.12	41.18	8.82	94.12	41.18	32.35	8.82	82.35
R2L	92.77	4.02	0.40	97.19	93.17	3.61	-	96.79
DOS	0.56	77.08√	0.05	77.70	0.56	99.10√	0.27	99.93
PRB	63.57	20.29	2.32	86.19	98.41	1.34	-	99.76
%far			0.22				0.20	

Table 6. Combined Intrusion & Anomaly Detection Data (cont.)

dataset ₉					dataset ₁₀			
	%tc	%a	%othr	%ttl	%tc	%a	%othr	%ttl
normal	99.08	0.91	na	na	99.40	0.59	na	na
back	98.64	1.36○	-	100.00	99.55	0.45○	-	100.00
buffer_overflow	40.00	40.00○	20.00	100.00	40.00	60.00○	-	100.00
loadmodule	66.67	33.33✓	-	100.00	88.89	11.11○	-	100.00
perl	66.67	-	-	66.67	100.00	-	-	100.00
rootkit	40.00	60.00○	-	100.00	100.00	-	-	100.00
ftp_write	100.00	-	-	100.00	100.00	-	-	100.00
warezclient	97.58	1.93○	-	99.52	96.62	3.38○	-	100.00
warezmaster	60.00	20.00○	20.00	100.00	60.00	-	40.00	100.00
guess_passwd	90.91	9.09○	-	100.00	90.91	9.09○	-	100.00
imap	100.00	-	-	100.00	100.00	-	-	100.00
land	100.00	-	-	100.00	100.00	-	-	100.00
portsweep	99.04	0.96○	-	100.00	99.04	0.96○	-	100.00
satan	99.05	0.95	-	100.00	98.42	0.63○	-	99.05
ipsweep	98.79	0.81○	-	99.60	100.00	-	-	100.00
nmap	89.13	10.87○	-	100.00	100.00	-	-	100.00
multihop	100.00	-	-	100.00	100.00	-	-	100.00
neptune	na	99.91✓	0.05	99.96	99.89	0.10○	-	99.99
phf	na	100.00✓	-	100.00	na	25.00	-	25.00
pod	na	100.00✓	-	100.00	na	100.00✓	-	100.00
teardrop	na	94.42✓	0.51	94.92	na	93.91✓	-	93.91
spy	na	-	-	-	na	50.00✓	-	50.00
smurf	na	100.00✓	-	100.00	na	100.00✓	-	100.00
U2R	61.76	32.35	2.94	97.06	88.24	11.76	-	100.00
R2L	94.38	4.02	0.40	98.80	93.57	4.02	0.80	98.39
DOS	0.56	99.40✓	0.02	99.98	27.92	72.06✓	-	99.98
PRB	98.41	1.47	-	99.88	99.14	0.49	-	99.63
%far			0.23				0.20	

dataset ₁₁					dataset ₁₂			
	%tc	%a	%othr	%ttl	%tc	%a	%othr	%ttl
normal	99.36	0.62	na	na	99.46	0.53	na	na
back	98.64	1.36○	-	100.00	98.87	1.13○	-	100.00
buffer_overflow	100.00	-	-	100.00	60.00	40.00○	-	100.00
loadmodule	77.78	22.22○	-	100.00	77.78	22.22○	-	100.00
perl	100.00	-	-	100.00	100.00	-	-	100.00
rootkit	90.00	10.00○	-	100.00	100.00	-	-	100.00
ftp_write	100.00	-	-	100.00	100.00	-	-	100.00
warezclient	98.55	0.97○	-	99.52	95.65	1.45	0.48	97.58
warezmaster	60.00	-	20.00	80.00	60.00	20.00○	-	80.00
guess_passwd	90.91	9.09○	-	100.00	90.91	9.09○	-	100.00
imap	91.67	-	-	91.67	91.67	8.33○	-	100.00
land	75.00	-	-	75.00	100.00	-	-	100.00
portsweep	99.04	0.96○	-	100.00	98.08	1.92○	-	100.00
satan	99.37	0.63○	-	100.00	97.47	2.53○	-	100.00
ipsweep	99.19	0.40○	-	99.60	99.19	0.40○	0.40	100.00
nmap	95.65	4.35○	-	100.00	93.48	6.52○	-	100.00
multihop	100.00	-	-	100.00	100.00	-	-	100.00
neptune	99.81	0.18○	-	100.00	99.81	0.18○	-	99.99
phf	100.00	-	-	100.00	100.00	-	-	100.00
pod	na	100.00✓	-	100.00	100.00	-	-	100.00
teardrop	na	86.29✓	1.52	87.82	97.46	2.54○	-	100.00
spy	na	-	-	-	na	-	-	-
smurf	na	100.00✓	-	100.00	na	100.00✓	-	100.00
U2R	91.18	8.82	-	100.00	88.24	11.76	-	100.00
R2L	96.39	1.20	0.40	97.99	93.98	2.41	0.40	96.79
DOS	27.89	72.07✓	0.01	99.97	28.21	71.79✓	-	100.00
PRB	99.02	0.86	-	99.88	97.92	1.96	0.12	100.00
%far			0.21				0.18	

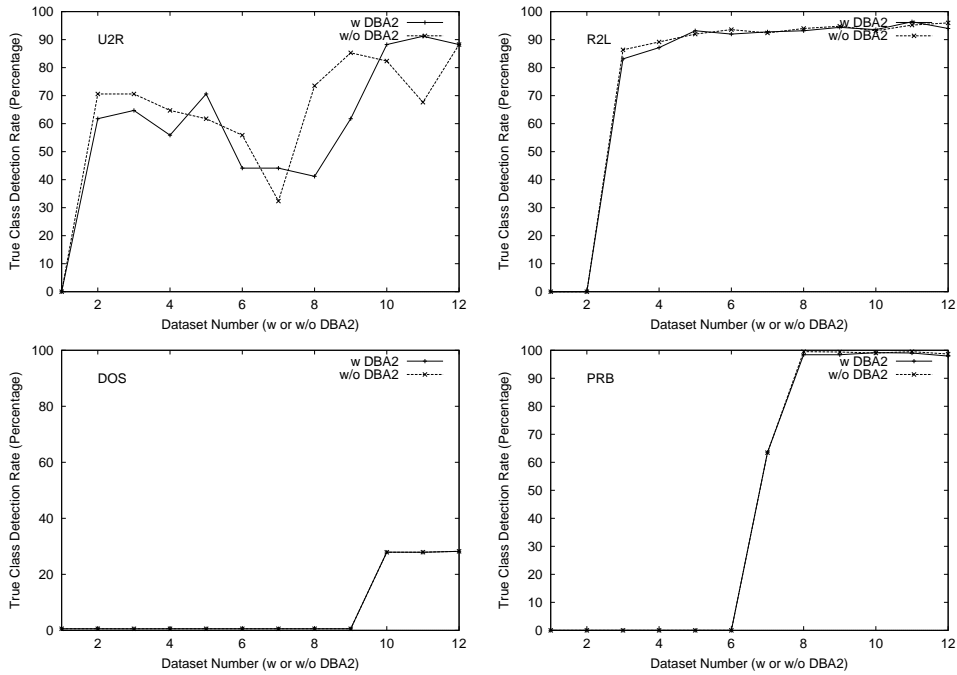


Fig. 4. Comparison of True Class Detection Rate (%tc) of Datasets w. and w/o DBA2

Table 7. Percentage of Significant True Anomaly Detections

Dataset	0	1	2	3	4	5	6	7	8	9	10	11	12
Anomaly Types	22	21	17	14	13	12	11	10	7	6	5	4	2
Significant	13	15	14	10	8	9	7	8	6	5	5	3	2
%	59	71	82	71	62	75	64	80	86	83	100	75	100

True Class Detection The true class detection rates of models learned with and without DBA2 are shown in Figure 4. The x-axis shows each dataset, ranging from dataset₁ to dataset₁₂ as explained in Section 5. We see that the curves for R2L, DOS and PRB are indistinguishable. The difference in U2R curves is reasonably small as well². This observation shows that the proposed DBA method does not deteriorate the effectiveness of detecting particular categories of known intrusions. Next, we examine the efficacy of our approach in detecting anomalies.

True Anomaly Detection The effectiveness of detecting true anomalies is shown in the second column of all sub-tables in Tables 4- 6. In all sub-tables, intrusion classes with *na* entries in the %tc columns are true anomalies, as they do not exist in the training data, and thus have no misuse rules generated for

² Note that there are only 34 U2R instances in the test data. Disagreements in just a few examples can make significant difference in %tc.

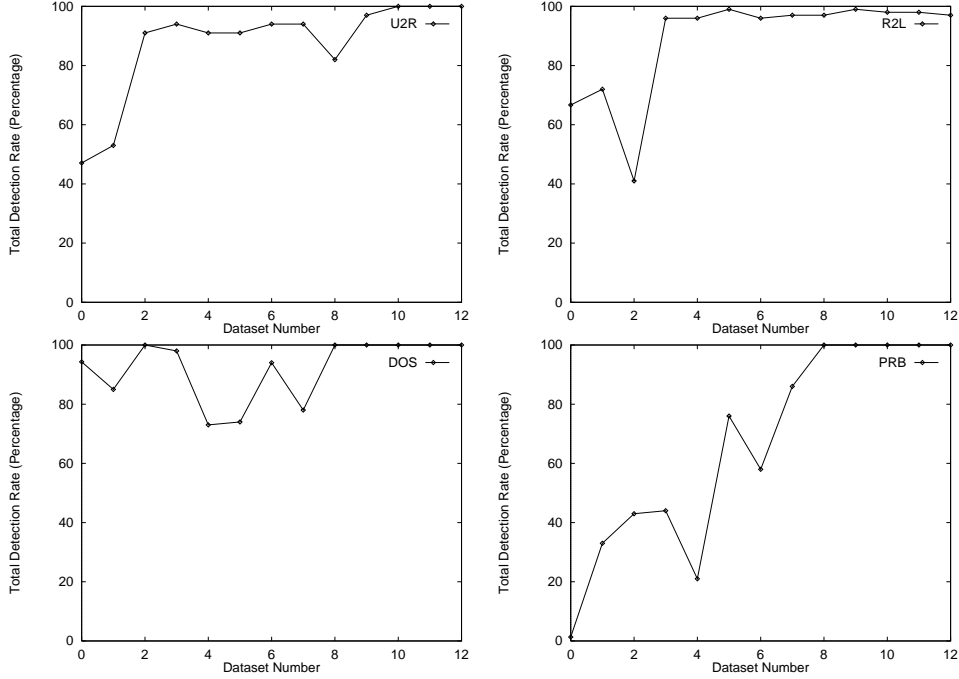
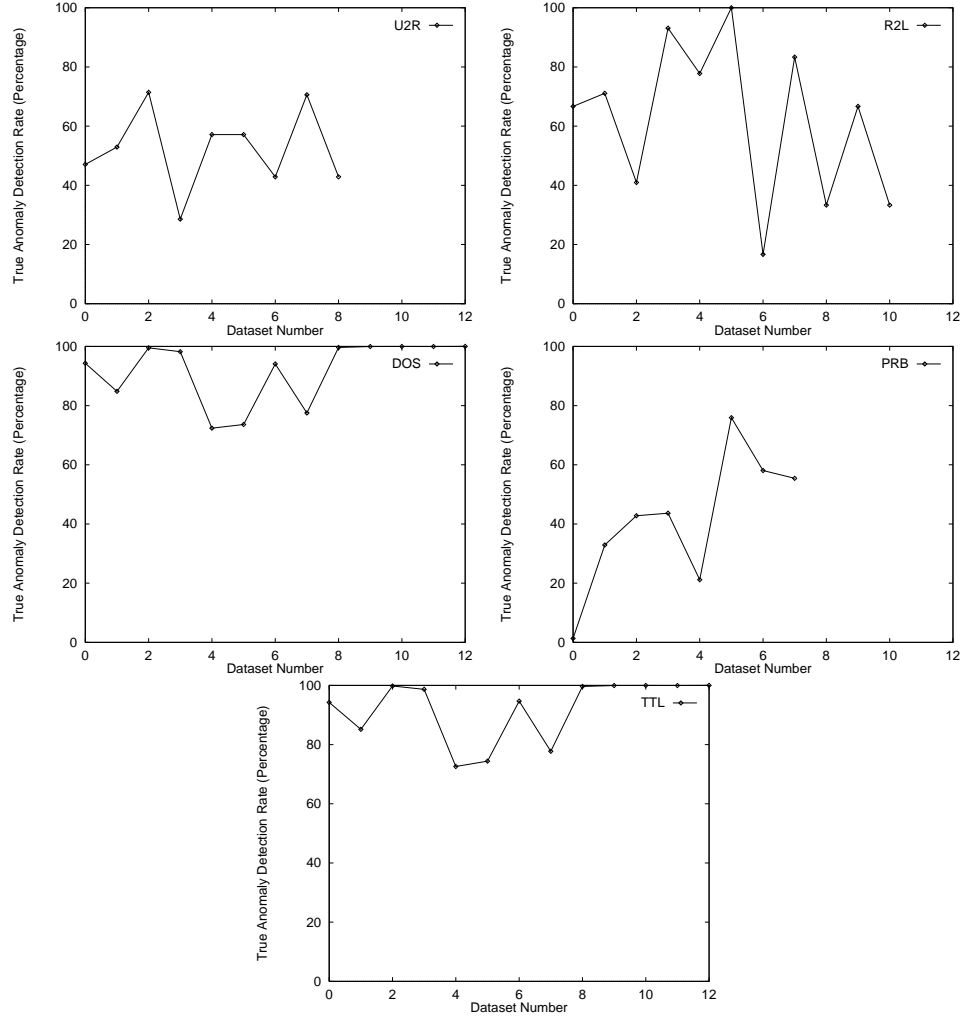


Fig. 5. Total Detection Rate ($\%ttl$)

them. In analysis, we consider an anomaly detection model to be *significant* for a particular detection class if $\%a \geq 50.00\%$. All significant results are highlighted by \checkmark . Across 12 experiment settings, there are 122 truly anomalous cases, and among them, 92 are significant; this is more than 75%. For each experiment setting, the percentage of cases that are significant is shown in Table 7; most are at least 70%. On the other hand, we found that 10 out of 122 (less than 9%) true anomaly cases are not detected. Upon closer inspection, most of them are *land* and *spy*. These two intrusions are usually detected by specific features, like “land_flag”. The datasets most likely do not contain values for these features that are needed to distinguish these intrusions from other known classes. The DBA2 anomalies generated from each dataset will thus not help in revealing the decision boundary (Section 8 discusses an alternative approach addressing this issue).

Next, we study the effectiveness of anomaly detection on different categories of true anomalies. We measure anomaly detection rate ($\%a$) of true anomalies in each intrusion category and all true anomalies (TTL). The results are presented in Table 8. As shown in the upper rightmost curve and the last row of the table under “TTL,” the true anomaly detection rate for all true anomalies remains relatively constant as we inject more clusters of intrusions. The curves for U2R, R2L and PRB categories are more bumpy than DOS because the anomaly detection model catches more in one category and fewer in the others.

Known Intrusions Detected As Anomalies It is interesting to determine if the proposed approach can prove effective in detecting un-classified known

Table 8. Percentage of True Anomalies Detected as Anomalies (%a)

Dataset	0	1	2	3	4	5	6
U2R	47.06	52.94	71.43	28.57	57.14	57.14	42.86
R2L	66.67	71.08	40.96	93.10	77.78	100.00	16.67
DOS	94.31	84.82	99.53	98.22	72.37	73.61	94.08
PRB	1.34	32.89	42.79	43.64	21.15	75.92	58.07
TTL	94.26	84.78	99.46	98.18	72.33	73.57	94.02

Dataset	7	8	9	10	11	12
U2R	70.59	42.86	na	na	na	na
R2L	83.33	33.33	66.67	33.33	0.00	0.00
DOS	77.52	99.65	99.96	99.98	99.95	100.00
PRB	55.44	na	na	na	na	na
TTL	77.46	99.65	99.96	99.97	99.95	100.00

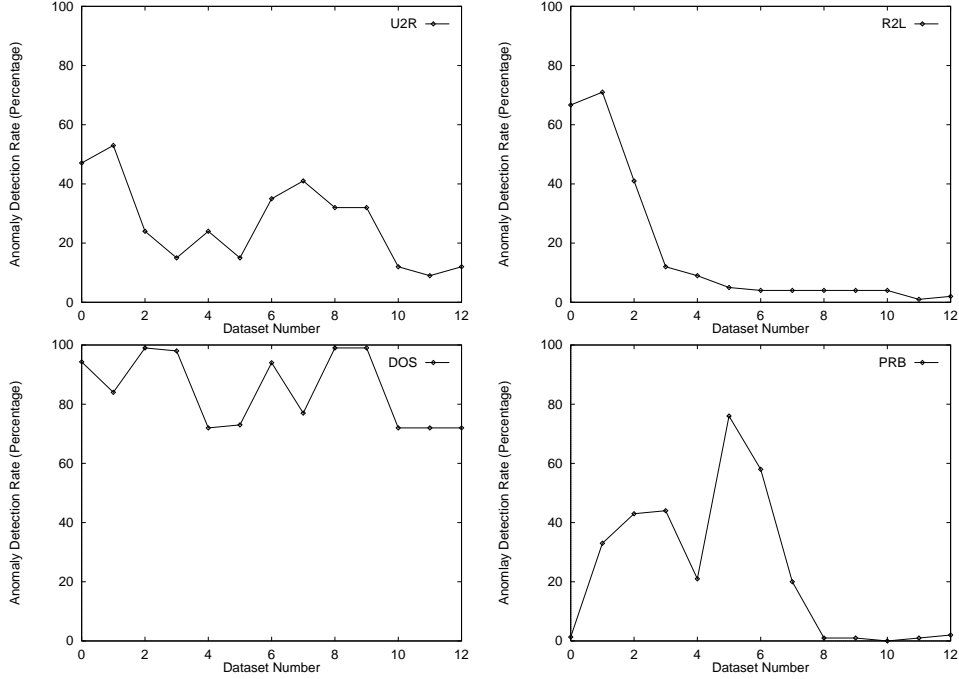


Fig. 6. Percentage of Known Intrusions and True Anomalies Detected as Anomalies (%)

intrusions as anomalies. We consider an anomaly detection method to *significantly compensate* for misuse detection if either the anomaly detection increases the total rate of detection to nearly 100% ($\%tc + \%a \simeq 100\%$, $\%tc < 100\%$) or $\%a \geq 0.25 \times \%tc$ when $\%tc$ is very low. In Tables 4 and 5, all significant compensations are highlighted by \odot . There are 88 cases that are candidates for compensation (i.e., $\%tc < 100\%$). Among them, 76 cases (or 86%) are significantly compensated by being detected as anomalies. In 5 of the remaining 12 cases, the intrusions are detected as some other intrusion, leaving no room for anomaly detection to provide any compensation. In Figure 5, we show the total percentage of detection ($\%ttl$) for all 4 categories of intrusions. As expected, there is a general trend of increase.

Overall Performance In the above discussion, we have covered the performance of true anomaly detection and misuse detection compensation. We now examine the combined overall performance of detecting both true anomalies *and* known intrusions. The results over all 4 intrusion categories are shown in the bottom of each sub-table of Tables 4 to 6, and the curves ($\%a$) are shown in Figure 6. As expected, there is a general trend of decrease in $\%a$ when the datasets are augmented with more clusters of intrusions. This is caused by the fact that we have learned misuse rules for more intrusions, leaving less room for these intrusions to be detected as anomalies. The shape of the anomaly detection rate curves is somewhat inversely related to their respective true class detection curves in Figure 4. This relationship is explained by the observation that $1 - \%tc$ is an indication of the amount of additional detection that anomaly detection

Table 9. Ruleset Size (Number of Rules)

dataset _i	1	2	3	4	5	6	7	8	9	10	11	12
w/o DBA2	7	33	54	58	52	58	65	80	97	98	94	106
w DBA2	183	203	240	136	131	25	220	227	238	284	256	281

can provide. These decreasing and inverse relationships apply throughout the U2R, R2L and DOS curves, and can be seen in the PRB curves with $x \in [6, 12]$. As we see in Figure 4, as more intrusion clusters are used to augment the *normal* data, the true class detection rates for both U2R and R2L increases and leave less room for anomaly detection to compensate. This explains the generally decreasing tendency of U2R and R2L %a curves. For DOS attacks, the true class detection rate only rises from 0 to 30% after dataset₉, and there is still sufficient room for compensation by anomaly detection — this explains the flatness of the DOS %a curve in Figure 6. For PRB, the rise in %tc takes place after dataset₆, which is also when we see the complimentary decrease in %a when $x \in [6, 12]$. The slight bumpiness of the U2R %a curve is due to the inverse bumpiness of U2R %tc curve in Figure 4. The slight bumpiness of the DOS and PRB curves are most likely caused by insufficient feature values available when learning a decision boundary for anomalies (See Section 8).

The last row of each sub-table within Tables 4 to 6 displays the false alarm rate of anomalous classification. All these values are uniformly below 0.40%. This confirms that nearly all detected anomalies are either true anomalies or known intrusions. Additionally, the %tc rates of *normal* connections (or *normal* correctly classified as *normal*), as shown in the first row of each sub-table, are over or near 99.00%. These observations show the utility of the anomaly detection approach in building highly precise models.

Effects of Cluster Ordering We performed two tests to verify that our results are not influenced by the order in which clusters are added to the training sets. One test is to reverse the cluster ordering as in the previous section, and the other one is a random ordering totally different from the original or second orderings. The results have confirmed that our results are indeed not influenced by cluster order.

6. Additional Issues

We now present our experimentation with different methods of anomaly generation for further verification of the proposed approach.

6.1. Necessity of Artificial Anomalies

To confirm the motivation of using artificial anomaly generation for classifying anomalies, we generated *unordered* rulesets from each dataset *without* any artificial anomalies and modified the default classification label to be *anomaly*³ as

³ *ordered* rulesets can not be used since the default rule is always used for a completely uncovered instance.

discussed in Section 2. We used different levels of “specificity” (different optimization levels “-0” from 0 to 2, and use very specific rule forms “-E”. Both “-0” and “-E” are parameters to run RIPPER). The results (not shown) indicate that these rulesets, in general, do not correctly detect many anomalies at all. The total number detected is very small and %*far* is very high (> 50%). Another observation is that rules learned from datasets with artificial anomalies are significantly more complex than rules trained without artificial anomalies, as shown in Table 9. The size differs by an average about 150 rules. Upon closer inspection, half of these extra rules are for *normal* and the other half are for *anomaly*. The reason is simple: artificial anomalies force RIPPER to discover very specific decision boundaries that are not revealed by known classes alone.

6.2. Different Experiment Settings

We have experimented with different amounts of injected artificial anomalies ($n = 1.5$ or 2). The general trend is that as we increase the injection amount, %*tc* of *normal* connections decreases slightly and %*far* increases slightly. In the worst case, %*tc* drops to slightly over 91% and %*far* increases to approximately 3%. The %*a* rate changes slightly as well, with some intrusion categories having greater %*a* rate, and some less, than when $n = 1$. On average, however, increased amounts of artificial anomalies is still as effective as with $n = 1$ (in the reported results). One explanation is that when the amount of injected artificial anomalies increases, there are more artificial anomalies than normal connections in the training data and the learning algorithm tends to generate more *anomaly* rules. In general, however, the proposed algorithm is not sensitive to the amount of artificial anomalies in the training data. However, it would be interesting to conduct a full range test on n .

We experimented with the use of other forms of RIPPER rulesets (*+freq* and *given*, both of which are *ordered* rulesets). As discussed by Fan et al. (Fan et al., 2000), ordered rulesets are more efficient in real-time evaluation. *+freq* rulesets classify connections in order of increasing frequency followed by *normal*, with a default classification of *anomaly*. For *given*, we used the following rule order: *normal*, *anomaly* and alphabetically ordered intrusion classes (essentially arbitrary). This *given* ruleset filters *normal* connections efficiently, as they are classified first. The results gathered for the use of a *+freq* ruleset are very close to the detailed results given for our *unordered* rulesets. It is interesting to observe that the *given* rulesets are similar to *unordered* rulesets at later datasets (when more than 3 clusters of intrusions are added to the normal data). However, in the first 2 datasets (dataset₁ and dataset₂), the anomaly detection is more likely to classify known intrusions as anomalies. This is due to the fact that anomaly rules appear before intrusion rules.

6.3. Other Artificial Anomaly Methods

To test the effectiveness of DBA2, we tested this method by randomly choosing a feature value to perturb, instead of biasing the choice to values that are more sparsely distributed as discussed in Section 2.2. We tested this method by randomizing any one randomly chosen feature, generating an artificial anomaly sample set with $n = 1$, and then learning *unordered* RIPPER rulesets. The

experiments were run on all datasets (0-12). Our results indicate that the $\%tc$ rates for each connection class (including *normal*) and $\%a$ are significantly lower than the respective DBA2 approach. The $\%tc$ rate for *normal* connections decreases to slightly above 51% in the worst case. The $\%tc$ rates for intrusions are also slightly reduced. The $\%far$ rate for all 13 experiments increased to slightly below 11%. In other words, the ruleset classifies more *normal* and known intrusions as anomalies. This verification method provides additional evidence for the effectiveness of the DBA2 proposal.

In Section 2.1, we discussed a random generation approach that produces artificial anomalies uniformly across the entire decision space. We produced a random sample for each dataset (the same size as the dataset) and trained over all of these datasets (0-12). Our results showed some detection of anomalies, but the amount was significantly less than was seen with the DBA2 approach. A close comparison of the respective rules showed that the difference in rules for the same connection class is minor. It is reasonable to deduce that the naively generated anomalies are not sufficient.

We experimented with the filtering method for DBA2 as proposed in Section 2.2. The generated artificial anomalies were filtered 3 times with a resulting reduction of between 1% and 3% each time. We did not see any significant improvement in performance using this method. We also experimented with filtering anomalies generated using the naive approach and observed that no artificial anomalies were removed at all. The main conclusion to be drawn from these filtering experiments is that most artificial anomalies are truly anomalous, and do not collide with known training data.

7. Related Work

SRI's IDES (Javitz and Valdes, 1991) measures abnormality of current system activity from the probability distributions of past activities. The activities they monitored are host events (e.g., CPU utilization and file accesses); in our work, we monitor network events. Forrest et al. (Forrest, Hofmeyr, Somayaji and Longstaff, 1996) record frequent subsequences of system calls that are used in the execution of a program (e.g., `sendmail`). Absence of subsequences in the current execution of the same program from the stored sequences constitutes a potential anomaly. Lane and Brodley (Lane and Brodley, 1998) used a similar approach but they focused on an incremental algorithm that updates the stored sequences and used data from UNIX shell commands. Lee (Lee, 1999), using a rule learning program, generated rules that predict the current system call based on a window of previous system calls. Abnormality is suspected when the predicted system call deviates from the actual system call. Ghosh and Schwartzbard (Ghosh and Schwartzbard, 1999) proposed using a neural network to learn a profile of normality. Similar to our approach, random behaviors are generated to represent abnormality for training purposes. Unlike our approach, each of their input features is a distance value from an exemplar sequence of BSM (SunSoft, 1995) events. This study is one of the first attempts in applying machine learning algorithms to network events for anomaly detection.

Algorithms for anomaly detection and misuse detection have traditionally been studied separately. In SRI's EMERALD (Neumann and Porras, 1999), anomaly and misuse detection algorithms are encased in separate system components, though their output responses are correlated to generate alarms by the

resolver. Ghosh and Schwartzbard (Ghosh and Schwartzbard, 1999) applied neural networks to both anomaly and misuse detection and compared their relative performance. One of our unique goals in this paper is to study the combination of anomaly and misuse detection in one model to improve overall performance.

We are not aware of closely related work in the generation of training data belonging to an unknown opposite class. Given unlabeled instances, Nigam et al. (Nigam, McCallum, Thrun and Mitchell, 1998) assigned labels to them using a classifier trained from labeled data and put them in the training set for another round of training. In a skewed distribution scenario, Kubat and Matwin (Kubat and Matwin, 1997) attempted to remove majority instances too close to and too far from the decision boundary. Maxion and Tan (Maxion and Tan, 2000) used conditional entropy to measure the regularity in the training set and have shown that it is easier to detect anomalies for data with high regularity. Lee and Xiang (Lee and Xiang, 2001) also applied entropy to determine how hard it is to learn a model of normality and abnormality.

8. Conclusion and Future Work

Recent hacker activity has made evident the importance of network-based intrusion detection. Anomaly detection of unknown intrusions is an important and difficult area of IDS. In this paper, we studied the problems of using artificial anomalies to detect unknown and known network intrusions. We proposed a distribution-based anomaly generation algorithm that has proven effective in building anomaly and combined misuse and anomaly detection models that successfully detect known and unknown intrusions.

One assumption of DBA2 is that each dimension (i.e., feature) can be treated individually. In other words, we examine and generate anomalies “dimension by dimension.” If all features have similar importance to the learner in formulating a model, anomalies generated using this assumption will be effective. A possible variation of the algorithm could consider multiple dimensions concurrently or give each dimension a different weight depending on its importance.

References

- Cohen, W. (1995), Fast effective rule induction, *in* ‘Proceedings of Twelfth International Conference on Machine Learning (ICML-95)’, Morgan Kaufman, pp. 115–123.
- Fan, W., Lee, W., Stolfo, S. and Miller, M. (2000), A multiple model approach for cost-sensitive intrusion detection, *in* ‘Proceedings of Eleventh European Conference on Machine Learning (ECML-00)’, Barcelona, Spain.
- Forrest, S., Hofmeyr, S. A., Somayaji, A. and Longstaff, T. A. (1996), A sense of self for UNIX processes, *in* ‘Proceedings of IEEE Symposium on Security and Privacy 1996’.
- Ghosh, A. K. and Schwartzbard, A. (1999), A study in using neural networks for anomaly and misuse detection, *in* ‘Proceedings of USENIX Security Symposium 1999’.
- Javitz, H. and Valdes, A. (1991), The SRI IDES statistical anomaly detector, *in* ‘Proceedings of IEEE Symposium on Security and Privacy’, p. 1991.
- Kubat, M. and Matwin, S. (1997), Addressing the curse of imbalanced training sets: One sided selection, *in* ‘Proceedings of Fourteenth International Conference on Machine Learning (ICML-97)’, Morgan Kaufmann, pp. 179–186.
- Lane, T. and Brodley, C. (1998), Approaches to online learning and concept drift for user identification in computer security, *in* ‘Proceedings of Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)’, pp. 259–263.

- Lee, W. (1999), A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems, PhD thesis, Columbia University.
- Lee, W. and Xiang, D. (2001), Information-theoretic measures for anomaly detection, *in* 'The 2001 IEEE Symposium on Security and Privacy', Oakland, CA.
- Maxion, R. A. and Tan, K. M. (2000), Benchmarking anomaly-based detection systems, *in* 'International Conference on Dependable Systems and Networks', pp. 623–630.
- Neumann, P. G. and Porras, P. A. (1999), Experiments with EMERALD to date, *in* 'Proceedings of 1999 USENIX Workshop on Intrusion Detection'.
- Nigam, K., McCallum, A., Thrun, S. and Mitchell, T. (1998), Learning to classify text from labeled and unlabeled documents, *in* 'Proceedings of Fifteenth National Conference on Artificial Intelligence (AAAI-98)'.
- SunSoft (1995), *SunSHIELD Basic Security Module Guide*, SunSoft, Mountain View, CA.

Author Biographies

insert photo

Wei Fan received a PhD degree in Computer Science from Columbia University in 2000. He is currently employed at IBM T.J.Watson Research Center. His research interests are in machine learning, data mining, database and security. Dr. Fan is the inventor of over ten patents in areas of intrusion detection, database and data mining.

insert photo

Salvatore J. Stolfo Professor of Computer Science at Columbia University. He received his Ph.D. from NYU Courant Institute in 1979 and has been on the faculty of Columbia ever since. He has published well over a hundred scientific publications in the areas of parallel computing, AI Knowledge-based systems, and most recently Data Mining, and Intrusion Detection systems. Dr. Stolfo co-developed the first Expert Database System in the early 1980's that was widely distributed to a large number of telephone wire centers around the nation. He has lead a project that developed the 1023-processor DADO parallel computer designed to accelerate knowledge-based and pattern directed inference systems. His most recent research has been devoted to distributed data mining systems with applications to fraud and intrusion detection in network information systems. Browse the URL <http://www.cs.columbia.edu/ids> for complete details. He recently co-chaired several workshops in the area of data mining, intrusion detection and the Digital Government and co-chaired the PC of the SIGKDD 2000 Conference. He presently co-directs the Digital Government Research Center, a joint research center between Columbia University and USC/ISI. He served as the Chairman of Computer Science and the Director of the Center for Advanced Technology at Columbia University. Recently, he was a member the Congressional Internet Caucus Advisory Committee, and Visa 3D Secure Authenticated Internet Payments Vendor Program. He is presently the Chief Science Advisor to System Detection Inc, a recently established start-up he co-founded to commercialize his DARPA-sponsored research in Data mining-based Fraud and Intrusion detection. He has been awarded ten patents in the areas of parallel computing and database inference, and a number of patents are pending in the area of internet privacy and security.

insert photo

Wenke Lee is an Assistant Professor in the College of Computing at Georgia Institute of Technology. His research interests include network security and data mining. He received his Ph.D. in Computer Science from Columbia University. He received a Best Paper Award (in applied research category) at the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99) in 1999, and a NSF CAREER Award in 2002.

insert photo

Philip K. Chan an Associate Professor of Computer Science at Florida Institute of Technology. His main research interests include scalable adaptive methods, machine learning, data mining, distributed and parallel computing, and intelligent systems. He received his PhD, MS, and BS in Computer Science from Columbia University, Vanderbilt University, and Southwest Texas State University respectively.

Correspondence and offprint requests to: Wei Fan, IBM T.J.Watson Research, Hawthorne, NY 10532, USA. Email: weifan@us.ibm.com