Implementing an Intrusion Detection System using a Decision Tree

Anubhavnidhi Abhashkumar

Roney Michael

***Abstract*—As the Internet becomes more and more accessible to people the world over, the realm of network security faces increasingly daunting problems. From the point of view of a defender, we now have to thwart the attempts of an increased number of malicious users; in the face of an attack, a larger consumer base left unserved turns out to be a larger margin of lost revenue. The value of being able to catch these attacks before they actually happen is thus going up with each passing day. This is where Intrusion Detection Systems (IDSs) come into play. This report describes over work in designing a Network IDS based on a canonical Decision Tree based approach. The system is trained and evaluated on pseudo-random subsets of the KDD-99 data set, which has been widely used to evaluate past and present day research in the domain.**

I. Introduction

As more and more computers get interconnected, the system in which a particular machine performs becomes increasingly chaotic. The combination of knowledge, time and certain disdainful facets of human nature invariably results in a situation where certain users of a computer system come under attack by others.

An Intrusion Detection System (IDS) is a collection of software and/or hardware components in any given system, whose objective is to correctly identify and possibly isolate malignant system activity. For the purposes of this report, the focus of our consideration is going to be a system of interconnected computer systems, for which the IDS specifically falls into the category of Network Intrusion Detection Systems (NIDSs).

In the most generic of terms, a NIDS operates by scanning input and/or output traffic for the network in which it operates and outputs a one-hot result classifying it as either normal traffic or any one of possibly many attacks. Several different categories of NIDS exist, some hinged on different ideas than others. Some of the more prominent ideas in developing these systems are: (i) Anomaly/Signature based; (ii) Passive/Active (iii)Stateful/Stateless.

Anomaly based NIDS operate based on the idea that the ambient traffic in a network collected over a period of time reflects the nature of the traffic that may be expected in the immediate future. A signature based system on the other hand works by identifying critical choke points in attacks and attempting to classify traffic based on the nature of the data at these points. An IDS may be classified as passive/active based on the action it takes when some malignant activity has been identified. Passive systems work by logging suspicious activity as and when detected, which may later be reviewed by a human operator to take the final call, while active systems may take actions such as account deactivation or disabling network access. Finally, the idea of stateful IDSs lend itself to the notion that an attack may be spread out over multiple possibly distributed sequences of data, while a stateless system operates only on a small current window of information at any given time. It is also important to note when the classifier actually acts; i.e., whether it is on-the-fly or after certain fixed intervals of time. The idea of a real-time classifier follows directly from that of a stateless one.

The focus of this paper is a signature based, real-time, passive and stateless NIDS based on based on Decision Trees.

II. Related Work

Intrusion detection techniques has secured a space in the network defense landscape alongside firewalls and virus protection systems. There are two primary methods of monitoring these are signature based and anomaly based. In [1] different approaches of IDS are analyzed. Some approach belongs to supervised method and some approach belongs to unsupervised method. Some of the approaches were Support Vector Machines, Artificial Neural network, Self-Organizing Map etc

Many different algorithms have been used for this purpose. For example [2] used support vector machine, [3] used naive bayes and [4] used bayesian network for the purpose of creating a good IDS.

Irrelevant and redundant features can often lead to models with poor performance. To substantiate the performance of machine learning based detectors that are trained on KDD 99 training data; [5] investigated the relevance of each feature in KDD 99 intrusion detection data sets, using information gain to determine the most discriminating features for each class.

[6] conducted a statistical analysis on this data set and found some issues which highly affected the performance of evaluated systems, and resulted in a very poor evaluation of anomaly detection approaches. To solve those issues, they proposed a new data set, NSL-KDD, which consists of selected records of the complete KDD data set and does not suffer from any of KDD's shortcomings.

III. Data Set

Evaluating the performance of a classification algorithm is a critical choke-point in determining how useful it might be. For the evaluation to be truly trustworthy, the data set has to conform to certain minimum requirements: (i) Statistically relevant, i.e., the data should mirror real world scenarios; (ii)Variables of mixed type; (iii) Should allow modeling using different perspectives for third party evaluation. For evaluating our NIDS, we initially chose the KDD-99 data set, one made popular through the 1999 SIGKDD intrusion detection contest.

The data set was based on that of a 1998 DARPA intrusion detection evaluation program which was prepared and managed by MIT Lincoln labs. It was collected over a period of

nine weeks through raw TCP dumps of a Local Area Network which was intended to simulate a typical US Air Force LAN. The training data consisted of nearly 5 million connection record and the test data, of nearly 2 million records. Each record indicated a connection, i.e, a sequence of TCP packets identified as corresponding to a single operation. Each record is labeled as either normal traffic or one specific type of a number of varieties of attacks.

Overall, there were 24 attack types which could be bucketed into 4 high-level categories:

- DoS: Denial of Service
- R2L: Remote to Local
- U2R: User to Root
- Probing: Surveillance, Port Scans, etc.

The data set has the additional characteristic that the test and training sets were drawn using different probability distributions, which is what one may expect from a real-world situation.

Each record consisted of 41 feature values and a label. The meaning and nature of each of these features is shown in Table I [5] .

For a uniformly sampled 10% of the data set, the different labels were found to have the following counts ,Table II.

Table II: Category of Attacks

Attack	#samples	Category
smurf	280790	dos
neptune	107201	dos
back	2203	dos
teardrop	979	dos
pod	264	dos
land	21	dos
normal	97277	normal
satan	1589	probe
ipsweep	1247	probe
portsweep	1040	probe
nmap	231	probe
warezclient	1020	r2l
guess_passwd	53	r2l
warezmaster	20	r2l
imap	12	r2l
ftp_write	8	r2l
multihop	7	r2l
phf	4	r2l
spy	2	r2l
buffer_overflow	30	u2r
rootkit	10	u2r
loadmodule	9	u2r
perl	3	u2r

The KDD 99 data set however notably suffered from a matter of redundancy. Though statistically viable, this drawback severely affects the data set's use as a measure of accurate predictions since an overwhelming majority of the records fell into one of just 3 classes out of a possible 25. In terms of numbers, the attacks in the training and test sets suffered from redundancies of 93.32% and 88.26% respectively. Taking into account the redundant normal traffic records as well, overall the training and test sets had redundancies of 78.05% and 75.15% . The total number of instances, redudndant instances and the reduction rate for training and test sets have been shown in Table III and Table IV respectively.

Table III: Statistics of redundancy in the KDD training set

	Original Records	Distinct Records	Reduction Rate
Attacks	3,925,650	262,178	93.32%
Normal	972,781	812,814	16.44%
Total	4,898,431	1,074,992	78.05%

Table IV: Statistics of redundancy in the KDD test set

	Original Records	Distinct Records	Reduction Rate
Attacks	250,436	29,378	88.26%
Normal	60,591	47,911	20.92%
Total	311,027	77,289	75.15%

For this reason, after careful consideration, we chose to go instead with a more recently published subset of the original data set, known as the NSL-KDD data set [6] . The elimination of the redundant records ensure two characteristics:

The performance of the classifier will not be biased by redundancy in the training set.

There will be no biased toward labels which possess a statistically higher frequency; it is expected to be independent of the distribution of the test set.

IV. Decision Trees

In working with real world data, we are often faced with quantities so huge that it becomes impractical to approach it from a traditional perception-based classification approach. Machine learning provides for a solution here through the use of supervised learning (classification) techniques, at times in conjunction with unsupervised learning (clustering) techniques. These methods are ever more important today given the fact that the size of data seems to be ever-increasing. Classification techniques have been employed with a considerable degree of success to aid authorities in the detection of credit-card theft, by academics to detect plagiarism and even in hostile aircraft detection on war-fronts.

A decision tree is a predictive classifier which makes a labeling call by branching off into one of possibly many child nodes (subtrees) at any of its internal nodes. A traversal of such a tree from the root to the extremities (leaves) provides us with the predicted label for that particular instance, associated with that particular leaf node. The concept behind decision trees are particularly alluring, given the fact that they lend themselves to being easily comprehensible to a typical human's understanding.

An extremely simplistic, but common example of the use of a decision tree is that of a guessing game as shown in Figure 1. Starting from the root node, any non-leaf node would pose a question about any particular feature of the instance to be classified. Depending on the value received as the answer (which may be considered as a boolean valued feature), we transit to one of its child nodes. This classification task iteratively continues until the point where we reach a leaf node, which corresponds to either a prediction or denotes a point of uncertainty. This algorithm may be further expounded to include a self-improvement methodology, where post-fact, when the classification result is actually known, if the prediction were to be wrong or undetermined, one could utilize the information to improve future predictions.

Table I: List of features with their descriptions and data types

Feature	Description	Type
duration	Duration of the connection	Continuous
protocol type	Connection protocol (e.g. rcp, udp)	Discrete
service	Destination service (e.g. telnet, ftp)	Discrete
flag	Status flag of the connection	Discrete
source bytes	Bytes sent from source to destination	Continuous
destination bytes	Bytes sent from destination to source	Continuous
land	1 if connection is from/to the same host/port; 0 otherwise	Discrete
wrong fragment	number of wrong fragments	Continuous
urgent	number of urgent packets	Continuous
hot	number of ẖotïndicators	Continuous
failed logins	number of failed logins	Continuous
logged in	1 if successfully logged in; 0 otherwise	Discrete
# compromised	number of c̈ompromisedc̈onditions	Continuous
root shell	1 if root shell is obtained; 0 otherwise	Continuous
su	attempted 1 if ”su root” command attempted;0 otherwise	Continuous
# root	number of ”root” accesses	Continuous
# file creations	number of file creation operations	Continuous
# shells	number of shell prompts	Continuous
# access files	number of operations on access control files	Continuous
# outbound cmds	number of outbound commands in an ftp session	Continuous
is hot login	1 if the login belongs to the ẖotlist; 0 otherwise	Discrete
is guest login	1 if the login is a g̈uestl̈ogin; 0 otherwise	Discrete
Count	number of connections to the same host as the current connection in the past two seconds	Continuous
srv count	number of connections to the same service as the current connection in the past two seconds	Continuous
serror rate	% of connections that have S̈YNërrors	Continuous
srv serror rate	% of connections that have S̈YNërrors	Continuous
rerror rate	% of connections that have R̈EJërrors	Continuous
srv rerror rate	% of connections that have R̈EJërrors	Continuous
same srv rate	% of connections to the same service	Continuous
diff srv rate	% of connections to different services	Continuous
srv diff host rate	% of connections to different hosts	Continuous
dst host count	count of connections having the same destination host	Continuous
dst host srv count	count of connections having the same destination host and using the same service	Continuous
dst host same srv rate	% of connections having the same destination host and using the same service	Continuous
dst host diff srv rate	% of different services on the current host	Continuous
dst host same src port rate	% of connections to the current host having the same src port	Continuous
dst host srv diff host rate	% of connections to the same service coming from different hosts	Continuous
dst host serror rate	% of connections to the current host that have an S0 error	Continuous
dst host srv serror rate	% of connections to the current host and specified service that have an S0 error	Continuous
dst host rerror rate	% of connections to the current host that have an RST error	Continuous
dst host srv rerror rate	% of connections to the current host and specified service that have an RST error	Continuous

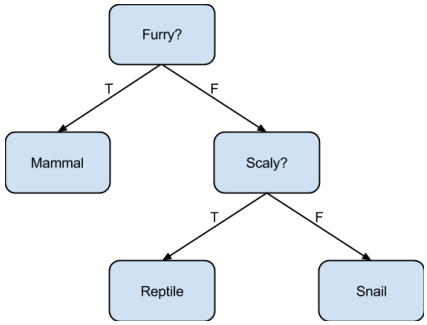

Figure 1: Sample Decision Tree

An important question which comes up in modeling a decision tree based classifier for a deterministic computer is which feature is to be tested at a particular level of nesting within it. Testing all possible features in any given order should of course yield the most accurate result (assuming the absence of outliers), but this increases the per instance classification cost by a considerable factor, which might turn out to be an unacceptable overhead, particularly for real time prediction requirements. Thus, identifying the most important question to be posed to a given instance is paramount. If we consider the classification of a large data set, then it is reasonable to say that the better the classifier performs, the more “pure” the leaf nodes will be in terms of the labels of the their instances. To define this “purity” as a quantifiable metric, we make use of the concept of entropy; the lower the entropy of the instance labels are at any given node, the “purer” the collection is. Taking this idea a step further, with each question posed, we expect the resultant data to have a better sense of homogeneity; i.e., the entropy of the parent set should be higher than that of the sum of the entropies of the result sets, which consequently implies that the greater this difference, the better our question. This is, in a basic sense, what we call as the information gain of a certain candidate feature over a particular subset of data, the measure by which we pick the best question.

At this point, it is important to note that there exist numerous variations of tree building algorithms in practice, each with it’s own pros and cons and there is still a considerable degree of debate in academia as to which is actually the best, given a particular problem. Now given that the object of our classification algorithm was to generate a tree which would be trained over a substantial amount of data, the major consideration (besides accuracy of course) was that of run time. Additionally, considering the vast amount of data this would have to start out with, updating the tree on a per instance level

```
'service' = ecr_i
|    'count' <= 20.500000
|    |    'src_bytes' <= 292.000000
|    |    |    'src_bytes' <= 25.000000  : ipsweep
|    |    |    'src_bytes' > 25.000000  : normal
|    |    'src_bytes' > 292.000000
|    |    |    'src_bytes' <= 1256.000000  : smurf
|    |    |    'src_bytes' > 1256.000000  : pod
```

Figure 2: Subset of our Decision Tree

is evidently excessive. These characteristics of the data led us to choose Dr. Ross Quinlan's Iterative Dichotomiser 3 as the basis of our NIDS. The working of ID3 as mentioned in is shown in the next paragraph.

ID3 learns a decision tree using the top-down approach, starting with the question "which attribute should be tested at the root of the tree?". For this, we have to do a statistical test on each attribute to find how well it alone classifies the training examples. The attribute which performs the best is chosen here as the test at root node. Descendants are created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e. down the branch corresponding to the example's value for this attribute). The pseudo code for a binary classification for ID3 from [7] is shown in Figure 3

For the purpose of Information Gain, we used the following formula,

$$InfoGain(D, S) = H_D(Y) – H_D(Y|S)$$

where $H_D(Y)$ represents entropy for class Y on data set D, $H_D(Y|S)$ represents conditional entropy of class Y on attribute S for data set D and $InfoGain(D, S)$ represents the Information Gain on data set D by splitting on feature S i.e. the expected reduction in entropy caused by partitioning the data set D according to this attribute S.

We also implemented the multi-class version of ID3 (We had to keep track of multiple classes instead of just 2). The main advantages of this algorithm is that it is computationally relatively simple, with run time increasing in a near-linear measure (O(n*log(n))); the disadvantage is that updation of the tree requires in essence, a complete rebuilding of the classifier.

V. Results and Discussion

A. Interpreting the Tree

As mentioned before, one of the major advantages of using decision trees is that the resultant classifier may easily be interpreted. Let us consider a subtree as shown in Figure 2 of the classifier obtained after running the algorithm on the NSL-KDD data set:

Where, 'ecr_i' and 'pod' stand for "echo reply ICMP" and "ping of death" respectively. This means that a connection instance of service type ecr_i with a count (number of connections to the same host as the current connection in the past two seconds) less than or equal to 20 (as it will be an integer value) and an src_byte length greater than 1256 will be flagged as a pod attack attempt.

B. Performance Metrics

For the purpose of this project we used Accuracy, Precision and Recall as our main performance metrics and to calculate them we had to calculate the values of True Positive(TP), False Positive(FP),True Negative(TN) and False Negative(FN).

TP represents those instances which are actually an attack and classified as an attack. FP represents those instances which are actually normal but classified as an attack. FN represents those instances which are actually an attack but classified as an normal. TN represents those instances which are actually normal and classified as an normal.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$
$$Precision = \frac{TP}{TP+FP} = \frac{TP}{PredictedPositive}$$
$$Recall = \frac{TP}{TP+FN} = \frac{TP}{Actualpositive}$$

In other words, Accuracy represents how many instances were correctly classified, Precision represents out of all the instances classified as an attack how many were actually an attack and Recall represents how many attacks were correctly classified (percentage of attacks caught). We got the following values for the NSL-KDD data set

- TP = 9118
- FP = 882
- TN = 8827
- FN = 3488
- Accuracy (on multi-class classification)= 69.61%
- Accuracy (on binary classification)= 80.39%
- Precision = 91.88%
- Recall = 72.33%

We got the following values for the KDD data set

- Accuracy = 99.97%
- Precision = 99.99%
- Recall = 99.99%

VI. Discussion

It is interesting to note that though we found a lot of research work based on the original KDD data set which boasted of accuracies of 99% and above, attributing them to superior, novel approaches, the canonical ID3 classifier gave the same (or even better) level of performance, which casts a certain amount of doubt on the validity of the earlier approaches. In addition to the reasonable level of accuracy, we were also able to obtain very good levels of precision (out of all the instances classified as an attack how many were actually an attack) and recall (proportion of attacks detected) in comparison to that advertised by some well regarded names in the industry. On the NSL-KDD data set we achieved a precision of 91.18% and a recall of 72.33%. With classification over all 25 categories of attacks, an accuracy of 69.61% was achieved, which was a bit lower that what we had expected, but with binary labeling of traffic (malicious vs normal, where all the attacks belong to malicious class), it went up to 80.39%. On

ID3($Examples$, $Target\_attribute$, $Attributes$)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
  - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
  - The decision attribute for *Root* $\leftarrow A$
  - For each possible value, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
    - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
    - If $Examples_{v_i}$ is empty
      - Then below this new branch add a leaf node with label = most common value of $Target\_attribute$ in *Examples*
      - Else below this new branch add the subtree ID3($Examples_{v_i}$, $Target\_attribute$, $Attributes - \{A\}$))
- End
- Return *Root*

* The best attribute is the one with highest *information gain*, as defined in Equation (3.4).

Figure 3: Pseudo code for ID3

the original KDD-99 data set, the results as expected were less than meaningful due to the ridiculous amount of redundancy; we were able to achieve 99.97% accuracy across all 25 labels, near perfect ($>$ 99.99%) precision and recall.

VII. Conclusion

We implemented an Intrusion detection system using an ID3 decision tree classifier. By observing the decision tree we found out what feature value or what set of feature values could cause an attack and what set of feature values exist in a normal traffic. We also showed why KDD99 data set is not reliable in evaluating an IDS system as they are dominated with a few class variables and have redundant instances. We evaluated our performance on the NSL-KDD data set. We ended up with a high precision value of 91.18%. Our recall was 72.33%, accuracy on the binary classification(normal vs malicious) was 80.39% and on multi-class classification was 69.61%. Because of our high precision value we can say that a very small amount (8.82%) of normal traffic is flagged as an attack in our IDS implementation.

References

[1] M. K. Lahre, M. T. dhar Diwan, and S. K. K. P. Agrawal, "Analyze different approaches for ids using kdd 99 data set," *International Journal on Recent and Innovation Trends in Computing and Communication, ISSN*, pp. 2321–8169.

[2] Y. B. Bhavsar and K. C. Waghmare, "Intrusion detection system using data mining technique: Support vector machine," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 3, 2013.

[3] U. Subramanian and H. S. Ong, "Analysis of the effect of clustering the training data in naive bayes classifier for anomaly network intrusion detection," *Journal of Advances in Computer Networks*, vol. 2, no. 1, 2014.

[4] H. Shirazi, A. Namadchian, and A. khalili Tehrani, "A combined anomaly base intrusion detection using memetic algorithm and bayesian networks," *differences*, vol. 16, p. 17, 2012.

[5] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: A feature relevance analysis on kdd 99 intrusion detection datasets," in *Proceedings of the third annual conference on privacy, security and trust*, 2005.

[6] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[7] T. M. Mitchell, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, 1997.