An Application of Machine Learning to Network Intrusion Detection

Chris Sinclair
Applied Research Laboratories
The University of Texas at Austin
sinclair@arlut.utexas.edu

Lyn Pierce
epierce@arlut.utexas.edu

Sara Matzner
matzner@arlut.utexas.edu

Abstract

Differentiating anomalous network activity from normal network traffic is difficult and tedious. A human analyst must search through vast amounts of data to find anomalous sequences of network connections. To support the analyst's job, we built an application which enhances domain knowledge with machine learning techniques to create rules for an intrusion detection expert system. We employ genetic algorithms and decision trees to automatically generate rules for classifying network connections. This paper describes the machine learning methodology and the applications employing this methodology.

1. Introduction

Existing intrusion detection systems rely heavily on human analysts to differentiate intrusive from non-intrusive network traffic. The large and growing amount of data confronts the analysts with an overwhelming task, making the automation of aspects of this task necessary. Whether complete automation is possible or even desirable is debatable. We describe the use of machine learning techniques which provide decision aids for the analysts and which automatically generate rules to be used for computer network intrusion detection.

The Applied Research Laboratories of the University of Texas at Austin (ARL:UT) has developed significant expertise in the area of machine learning through internal research and development. This research has been applied to a number of applications in which machine learning techniques, such as generation of finite state machines and pattern matching, have been used[5][6]. The Network Exploitation Detection Analyst Assistant (NEDAA) is one such application, combining artificial intelligence rule generation with a classic expert system as an enhancement for intrusion detection systems (IDS).[1] In this paper we give an overview of machine learning techniques used and we describe some of the successes and problems encountered in applying these techniques to computer network intrusion detection.

Our application layers machine learning techniques onto an existing network-based IDS deployed to protect military subnetworks. On each military subnetwork there is a probe that filters and logs network traffic to a central database[8]. A rule set is used to analyze archived data for intrusive patterns. The pattern matching has traditionally been simple, looking for exploitive activity such as connections from certain IP addresses with histories of intrusive behavior. However, an intrusion into a computer network can be more complex, with the complexity being both spatial and temporal. An example of this type of intrusion is a 'low and slow' attack consisting of intrusive behavior over hours, days or weeks that may originate from multiple network sources. Machine learning can be applied to this problem to extend human pattern recognition. Automated techniques are ideal for this application because they can monitor and correlate vast numbers of intrusive signatures.

[1] Contracts N00039-0051, Task Order No. 0293 and Task Order No. 0273 and ARL:UT IR&D Programs No. 0000859 and 0000875.

2. Machine Learning Techniques

Our current implementation of NEDAA contains rule generation modules that interface with two ARL:UT artificial intelligence (AI) software packages: a genetic algorithm tool set and a decision tree generator. These modules can be customized for specific applications and data sources. The choice of AI techniques employed stemmed from our previous experience with genetic algorithms, and literature research into other applicable methods. The genetic algorithm software package was a logical platform from which to tackle the difficult problem of intrusion detection. The use of decision trees for rule generation was made to provide a deterministic alternative to genetic algorithms.

VulcanRG, the machine learning component of the NEDAA system, generates rules for compilation into intrusion detection systems. These rules are generated by the genetic algorithm and by decision tree packages developed at ARL:UT. We currently use VulcanRG to generate rules for one deployed IDS and one experimental system. Many of the examples presented in this paper are derived from actual runs of the machine learning components of NEDAA.

2.1. Genetic Algorithms

Genetic algorithms are a family of problem-solving techniques based on evolution and natural selection. They are essentially a type of search algorithm, and as such, can be used to solve a wide variety of problems. This section gives a brief overview of genetic algorithms.

The goal of genetic algorithms is to create optimal solutions to problems. Potential solutions to the problem to be solved are encoded as sequences of bits, characters or numbers. The unit of encoding (usually a single bit in traditional genetic algorithms) is called a gene, and the encoded sequence is called a chromosome. The genetic algorithm begins with a set (population) of these chromosomes and an evaluation function that measures the fitness of each chromosome, i.e. the 'goodness' of the problem solution represented by the chromosome. It uses reproduction (one of several operators collectively called crossover operators) and mutation (the spontaneous alteration of a single gene) to create new solutions, which are then evaluated. The selection of chromosomes for survival and recombination is biased toward the fittest individuals. The recombination/evaluation sequence is iterated many times, and if the problem is well-constructed, strong solutions gradually emerge.

The genetic algorithm package we have developed is a generalization of the classic genetic algorithm. Our genetic algorithm does not mandate the encoding of solutions into low-level chromosomes. If crossover and mutation operators can be imposed on the solutions themselves, domain specific information can be used to expedite the search. If a low-level encoding was required in such cases, this domain specific knowledge would be unusable. In cases where a crossover or mutation operator cannot be imposed on the space of solutions, a classic genetic algorithm (with encoding to chromosomes) is used.

2.2. Decision Trees

Decision trees are structures used to classify data with common attributes. Each decision tree represents a rule which categorizes data according to these attributes. A decision tree consists of *nodes*, *leaves*, and *edges*. A node of a decision tree specifies an attribute by which the data is to be partitioned. Each node has a number of edges which are labeled according to a possible value of the attribute in the parent node. An edge connects either two nodes or a node and a leaf. Leaves are labeled with a decision value for categorization of the data.

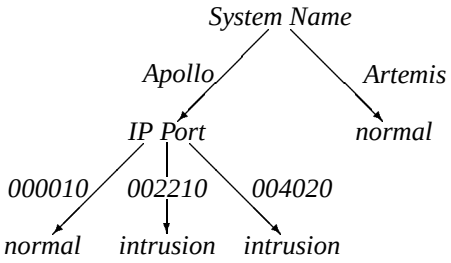
Example 1 *A decision tree to detect intrusive behavior based on the data in Table 1*

In this example IP Port, *and* System Name *label the nodes,* intrusion *and* normal *label the leaves, and the labeled arrows are the edges. The generated decision tree is shown in Figure 1.*

Table 1. Example Intrusion Data

IP Port	*System Name*	*category*
004020	*Artemis*	*normal*
004020	*Apollo*	*intrusion*
002210	*Artemis*	*normal*
002210	*Apollo*	*intrusion*
000010	*Artemis*	*normal*
000010	*Apollo*	*normal*

Figure 1. Example Intrusion Decision Tree

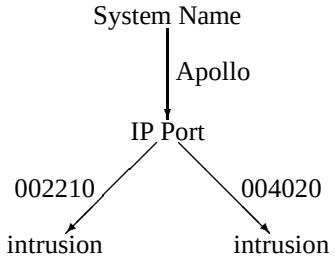


This decision tree could be turned into the following rule (in C++ form):

```
if(System Name == Artemis){
  intrusion = false;
}
else if(System Name == Apollo){
  if(IP Port == 002210){
    intrusion = true;
  }
  else if(IP Port == 004020){
    intrusion = true;
  }
  else if(IP Port == 000010){
    intrusion = false;
  }
}
```

We use Quinlan's[7] ID3 algorithm to construct decision trees from structured data (such as the data in Table 1). The ID3 algorithm uses information theoretic precepts to create efficient decision trees. Given a structured data set, a list of attributes describing each data element, and a set of categories to partition the data into, the ID3 algorithm determines which attribute most accurately categorizes the data. A node is established and labeled by this attribute. The edges coming from this node are labeled with the possible values of the partitioning attribute. The data set is then divided into subsets by the values of this attribute. If a subset is completely categorized, then the edge terminates in a leaf labeled by the categorization. Otherwise the subset is subdivided further by creating a new node and repeating this process recursively.

Figure 2. Pruned Decision Tree



Decision trees constructed by the ID3 algorithm are based on the *training set* used to construct them. In order for a decision tree to generalize the information learned, the decision tree must be *pruned*. Pruning replaces certain nodes with leaves. A simple example of pruning involves removing all nodes and leaves which terminate in a default category. In Figure 2 we show the tree of Figure 1 after it has been pruned. All connections are assumed to be normal if they are not classified as intrusions. The pruned decision tree can be turned into the following code:

```
intrusion = false;

if(System Name == Apollo){
  if((IP Port==002210)||(IP Port==004020)){
    intrusion = true;
  }
}
```

3. Machine Learning applied to Intrusion Detection

The NEDAA machine learning approach uses analyst-created training sets for rule development and analyst decision support. In the current implementation the training information is comprised of database views queried from the archived network events (Table 3). From this training data,

the machine learning modules generate rules of the form:

$$\text{if} < condition > \text{then} < action >$$

These rules can be compiled into the expert system for intrusive event detection, or used to simplify the analyst's task by summarizing large sets of training data into simple rule sets. It is important to note that the quality of the synthesized rules depends on the quality of the training set. Any classification errors in the training set will propagate into the resulting rule set.

Table 2. Example Training Data

Source IP	Dest IP	Source Port	Dest Port	Protocol	Intrusion
123.202.72.109	225.142.187.12	001360	000080	IP	true
123.202.72.109	225.142.187.12	001425	000080	IP	true
123.202.72.109	225.142.187.12	001488	000080	IP	true
123.202.72.109	225.142.187.12	001559	000080	IP	true
123.202.72.109	225.142.187.12	001624	000080	IP	true
123.202.72.109	225.142.187.12	002156	000080	IP	true
123.202.72.109	225.142.187.12	002158	000080	IP	true
225.142.147.75	150.216.191.119	001624	000080	IP	true
225.142.187.19	125.250.187.19	004207	000025	IP	true
233.167.15.65	225.142.187.12	004607	000025	IP	false
233.167.15.65	225.142.187.12	004690	000025	IP	false
139.61.51.70	225.142.187.12	001052	000021	IP	false
142.142.5.113	225.142.187.12	001572	000080	IP	false
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	false

3.1. Applying Genetic Algorithms

In the current version of NEDAA, we use genetic algorithms to evolve simple rules for monitoring network traffic. These rules are simple single-connection patterns for differentiating normal from abnormal network connections. The rules are evolved by creating patterns which match a set of anomalous connections and a set of normal connections (as reported by an analyst). These patterns are used as the firing conditions of rules of the form if $< pattern\_matched >$ then $< generate\_alert >$.[2] The goal is to develop rules which match only the anomalous connections. Such rules can then be used to filter historical records and new connections so that the analyst can concentrate on those which are suspicious.

The patterns created by the genetic algorithm corresponds to the format of incoming connections and are combinations of specific connection attribute values and 'wild cards'. Examples of attributes used in the current version of NEDAA include: source IP address, destination IP address, source IP port, destination IP port, and network protocol.

The initial population (see section 2.1) is comprised of random rules; an example is shown in Table 3. The chromosome for a rule is comprised of 29 genes: 8 for source IP (2 hexadecimal digits per address field), 8 for destination IP, 6 for source port, 6 for destination port, and 1 for protocol. Each gene can be either a specific numeric value in the appropriate range for the field or a wild card. The chromosome for the rule in Table 3 is shown in Figure 3.[3] When a rule is used to filter connections, each connection is converted to a 29-field format corresponding to the gene structure of a rule as described above. A rule matches a connection if and only if every non-wildcard gene in the rule matches the corresponding field in the connection.

Table 3. Sample rule

Attribute	Value
Source IP	42.22.e5.bc (66.34.229.188)
Dest IP	15.b*.6e.76 (21.176+?.110.118)
Source port	047051
Dest port	912320
Protocol	TCP

Figure 3. Chromosome for rule in Table 3

$$(4, 2, 2, 2, 14, 5, 11, 12, 1, 5, 11, -1, 6, 14,$$
$$7, 6, 0, 4, 7, 0, 5, 1, 9, 1, 2, 3, 2, 0, 17)$$

The evolutionary process of the genetic algorithm requires some form of fitness measure on individual population members. In NEDAA, this fitness measure is based on the actual performance of each rule on a pre-classified data set. Each rule is used to filter a data set comprised of connections marked as either anomalous or normal by an analyst, and the fitness function rewards partial matches of training connections that have been designated as anomalous. If a rule completely matches an anomalous connection

[2] As there is a clear 1-1 correspondence between rules and patterns, we will use the terms interchangeably for the remainder of this section.

[3] The value of -1 in the 12th field represents a wild card.

it is awarded a bonus; if it matches a normal connection it is penalized. As a result, succeeding populations are biased toward rules that match intrusive connections only. After a certain number of generations, the genetic algorithm is stopped, and the best unique rules are selected.

The classic genetic algorithm described earlier tends to converge on a single 'best' solution to any given problem. In the case of a rule set, however, it is generally not sufficient to find a single rule. Multiple rules are generally needed to identify unrelated types of anomalies; several 'good' rules are more effective than a single 'best' one. In mathematical terms, this requirement translates to the concept of finding *local maxima* as opposed to the *global maximum* of the fitness function. Genetic algorithms evolve solutions which maximize (or minimize) the fitness function. A traditional genetic algorithm will attempt to find a global maximum of the fitness function, and will continue until all solutions in the population have converged to this maximum. In contrast, the problem of discovering multiple rules to filter incoming connections based on different criteria is essentially that of discovering multiple local maxima of the fitness function.

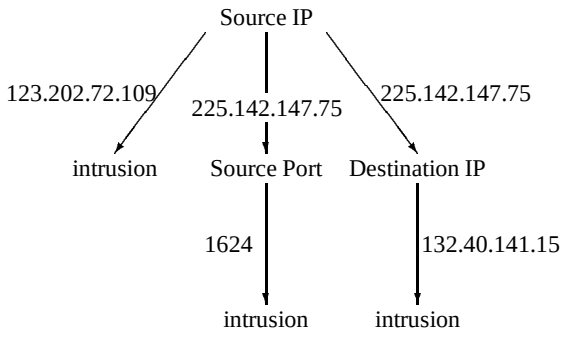
In order to find local maxima of the fitness function, and hence multiple rules, we employed *niching* techniques. Conceptually, niching in genetic algorithms is similar to that in nature; different species that share an environment generally inhabit different niches in order to exploit resources and minimize competition. In genetic algorithms, niching strategies attempt to create subpopulations which converge on local maxima. The two standard ways of niching are *sharing* and *crowding*[4]. Sharing degrades the fitness of solutions based on the number of other solutions which are nearby (similar). When the fitness is degraded in this manner, overcrowded niches become less hospitable, forcing solutions to other local maxima which may be less populated. In crowding, solutions which are generated by the crossover of two 'parent' solutions replace the nearest (most similar) solutions in the population.

Both sharing and crowding use the concept of *nearness*, or similarity, in order to maintain population diversity. A distance metric must be imposed either on the space of solutions or on the space of chromosomes in order to use either of these niching methods. Unless a domain specific distance metric is determined for the problem, the *Hamming distance* is used[4]. The Hamming distance between two chromosomes is the number of genes which differ between the two chromosomes. We use the Hamming distance and a variation of crowding in order to generate a diverse rule set.

3.2. Applying Decision Trees

We use the ID3 algorithm to create decision trees which classify connections based on the attributes listed in Table 3. The generated decision tree can be pruned to determine connections which have similar attributes to those in Table 3. In this way decision trees generalize information learned during their construction. A pruned decision tree generated by the ID3 algorithm based on data in Table 3 is shown in Figure 4.

Figure 4. Intrusion Detection Decision Tree



The rules produced by the decision trees are of a slightly different nature than those produced by genetic algorithms. The genetic algorithms use niching to create a population of unique rules; ideally each rule at its own local maxima. Decision trees on the other hand create a single rule with a number of different clauses. Each clause is equivalent to a single rule in the genetic algorithm population. Each element of training data falls into one of the clauses in the

decision tree and hence is represented in the final rule. The 'completeness' of the final rule, with respect to the training data, is an advantage that decision trees have over genetic algorithms, and dispenses with the need for niching in the decision tree component.

The rules developed by the decision tree component, like the genetic algorithm rules, are sensor level rules. These rules are used to filter single connections. Currently, the generated decision trees provide an elaboration of the 'Hot IP' list; a list which enumerates IP addresses with previous anomalous activity. This extension of the 'Hot IP' list allows an intrusion detection system to differentiate between normal and anomalous activity from a Hot IP address.

4. Future Plans

The goal of machine learning as applied to network intrusion detection is to generate a minimal rule set which can detect intrusion signatures generalized from previous activity. We want a minimal number of rules for rapid response and efficiency in the expert system. Rule complexity must mirror complexity of attacks; as hackers become more skilled our AI techniques need to create correspondingly complex rules. Our primary near-term goal is to extend the machine learning components to correlate and filter sets of connections as opposed to single connections. We will combine connection filtering with other information the IDS records (events, strings matched etc.) to create complex rules based on data annotated by analysts. More complex rules will look for connection patterns which are extensive in both space and time. Ideally these rules will be able to detect the 'low and slow' attack.

There are many ways to build rules that are based on a number of connections, events, etc. One way is to create rules which can cause other rules to be activated. This method, known as *rule chaining*, allows complex sequences of events to be detected[3]. Given a number of rules of the form $\{\text{if} < predicate > \text{then} < action >\}$, the execution of one rule's action during a cycle may trigger the successful evaluation of another rule's predicate on subsequent cycles. In this way rules may communicate with each other to detect complex behavior. Incoming connections may activate certain rules, which in turn may activate other rules. This process may continue indefinitely until a message triggers an alarm to the intrusion detection system, or until no new messages are created. We are investigating the alteration of the genetic algorithm component to create rules which chain in this manner.

Enhancements to the current decision tree module would be useful for extending their current utility. Decision trees work best for attributes with a small number of values. Planned enhancements would allow for the use of many-valued attributes[1]. The ID3 algorithm builds decision trees from an annotated data set. If the data set is augmented, a new decision tree must be built to encompass the changes. Building a decision tree is computationally intensive. In order to avoid this computation new algorithms have been developed to update existing decision trees based on new information[2]. This allows us to build scalable decision trees, and thus continually refine the rule set as new information becomes known to the analyst. We will investigate scalable decision tree construction as an enhancement to our current system.

We are also looking to employ the decision tree module to identify anomalous sequences of network events. Decision trees can be used to cluster network events into similar categories based on common attributes. Intrusive sequences of network events would be associated with sequences of corresponding clusters. By labeling these sequences as intrusive, we can generalize the specific intrusion sequences to encompass similar sequences.

The decision tree module currently generates decision trees by maximizing the information gain ratio at each level of the tree. This produces a decision tree which attempts to accurately differentiate network events based on their common attributes. By replacing the information gain ratio with a distance function on the set of training data, the resulting decision tree would partition the training set into subsets defined by similarity. By using a distance function with a scalable decision tree builder, one could create a decision

tree which clusters the set of archived network events. This decision tree would contain almost all of the information of the archived data, but in a more manageable and compact format.

This decision tree snapshot of the archived network traffic could be used to create rules which chain to detect complex intrusions. Each network event would fall into a unique partition determined by the decision tree. Sequences of network events would be mapped to sequences of corresponding partitions. Since all elements in a given partition are similar, a sequence of partitions could represent a number of related (but distinct) sequences of network events. By building rules based on sequences of partitions generated by known intrusion signatures, rules could be built which detect those intrusions, and other similar sequences of network events.

The decision trees built to cluster data have additional value beyond rule generation for intrusion detection. These decision trees represent snapshots of the archived data, which can be used when lightweight approximations of the archived data are necessary.

In addition to enhancing our current suite of machine learning techniques, we also intend to research other artificial intelligence methods applicable to intrusion detection. Methods such as neural nets, and statistical methods may have utility in expanding our capabilities. The complexity of attacks that we can detect will improve as our machine learning techniques improve.

5. Conclusion

We have adapted existing machine learning applications to develop rules for a deployed IDS. The rule generation component of NEDAA is layered onto an expert system that enhances the ability of the IDS to filter anomalous connections. The current machine learning approach uses genetic algorithms and decision trees. The rules we have developed and deployed differentiate anomalous connections from normal network connections. Planned near-term improvements will allow for more complex rule development. Created rule sets are to be evaluated against known data sets such as training data from the DARPA Intrusion Detection Evaluation. Preliminary analysis of the DARPA Intrusion Detection Evaluation Data using our machine learning components has yielded patterns in the data set attributed to the contrived nature of the training data.

The main result of the presented material is the production of rules for compilation into the expert system. We are pursuing the creation of rules to detect complex network intrusions to maximize the utility of the expert system, and to produce a dynamic rule base capable of detecting new attack signatures.

References

[1] Roger Gallion, Daniel C. St. Clair, Chaman L. Sabharwal, W.E. Bond (1993). "Dynamic ID3: A Symbolic Learning Algorithm for Many-Valued Attribute Domains," *SAC*: 14-20.

[2] Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, Wei-Yin Loh (1999). "BOAT - Optimistic Decision Tree Construction," To appear in *Proceedings of 1999 SIGMOD Conference.*

[3] David E. Goldberg (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA.

[4] Brad L. Miller, Michael J. Shaw (1996). "Genetic Algorithms with Dynamic Niche Sharing for Multimodal Function Optimization," *IEEE International Conference on Evolutionary Computation*: 786-791.

[5] Lyn Pierce, Stan Young (1998). "YAGATS: A Toolset for Genetic Manipulation of Finite-State Machines," Applied Research Laboratories Technical Report No. 99-1 (ARL-TD-99-1), Applied Research Laboratories, The University of Texas at Austin.

[6] Lyn Pierce, Chris Sinclair (1999). "YAGATS IR&D Report," Applied Research Laboratories Technical Report No. 98-1 (ARL-TD-98-1), Applied Research Laboratories, The University of Texas at Austin.

[7] J. Ross Quinlan (1993). *C4.5 Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo, CA.

[8] Lane B. Warshaw, Lance Obermeyer, Daniel P. Miranker, Sara P. Matzner (1999). "VenusIDS: An Active Database Component for Intrusion Detection," Submitted to 1999 Annual Computer Security Applications Conference.