Automated Discovery of Concise Predictive Rules for Intrusion Detection[*]

Guy Helmer, Johnny Wong, Vasant Honavar, and Les Miller
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, IA
{ghelmer,wong,honavar,lmiller}@cs.iastate.edu

Abstract

This paper details an essential component of a multi-agent distributed knowledge network system for intrusion detection. We describe a distributed intrusion detection architecture, complete with a data warehouse and mobile and static agents for distributed problem-solving to facilitate building, monitoring, and analyzing global, spatio-temporal views of intrusions on large distributed systems. An agent for the intrusion detection system, which uses a machine learning approach to automated discovery of concise rules from system call traces, is described.

We use a feature vector representation to describe the system calls executed by privileged processes. The feature vectors are labeled as good or bad depending on whether or not they were executed during an observed intrusion. A rule learning algorithm is then used to induce rules that can be used to monitor the system and detect potential intrusions. We study the performance of the rule learning algorithm on this task with and without feature subset selection using a genetic algorithm. Feature subset selection is shown to significantly reduce the number of features used while not adversely affecting the accuracy of predictions.

Keywords

Intrusion detection, machine learning, feature subset selection

[*] This work supported by the Department of Defense under contract MDA 904-98-C-A887.

1 Introduction

Detecting intrusions [Denning87] against distributed computing systems is a difficult problem. Manually detecting intrusions in a distributed system requires a tremendous amount of effort and is prone to error. Automating the process of detecting intrusions is a topic of earnest research by several groups, including ours.

Our intrusion detection system is based on the concept of distributed knowledge networks (Honavar, Miller, and Wong, 1998) and data warehouse techniques. Distributed knowledge networks use agents for information retrieval and extraction, data transformation and knowledge discovery. Data warehouse technologies are used for data and knowledge organization and assimilation from heterogeneous physically distributed data and knowledge sources. This approach provides a modular and extensible approach to building complex and adaptive information systems for knowledge-intensive applications. Such distributed knowledge networks offer a natural framework for design and implementation of systems for monitoring distributed systems for intrusions (including concerted attacks spread over space and time) and the initiation of suitable countermeasures.

Our current implementation includes:

- Mobile and static data gathering agents that collect system logs and audit data and render them into a common format;
- Low level agents that monitor and classify ongoing activities, classify events, and pass on this information to higher level agents and to each other; and
- Data mining agents that use machine learning to acquire predictive rules for intrusion detection from system logs and audit data.

The higher level agents would provide a high-level intrusion detector, able to analyze attacks over the whole system, execute countermeasures, and support the system administration in their pursuit of attackers. The data warehouse knowledge provided by system of agents could be used to help systems management staff learn misuse patterns, understand attacks, and support offensive & defensive actions. Our intrusion detection system has been implemented using Java and ObjectSpace's Voyager mobile agent facility [ObjectSpace97].

In this paper, we sketch the design of our distributed intrusion detection system and then explain our research in the application of artificial intelligence to the problem of identifying misuse of privileged programs. The Computer Immunology project [Forrest96] and the Java Agents for Meta-Learning project [Lee98] explored the use of system call traces from privileged programs to detect intrusions. We use a feature vector approach to describe the system calls executed by a privileged program and then apply feature subset selection using a genetic algorithm. We show that our feature vector representation works well for automated knowledge discovery using rule learning, and that feature subset selection reduces the number of features necessary in the feature vector and improves the accuracy of the results by eliminating extraneous features.

2 Related Work

Projects closely related to our intrusion detection project include CIDF, DIDS, Computer Immunology, JAM, and AAFID.

The Common Intrusion Detection Framework (CIDF) [Reilly98] resembles our project's architecture. CIDF is a standard proposed by a group including the Information Technology Office of the Defense Advanced Research Projects Agency, University of California-Davis, Information Sciences Institute, Odyssey Research, and others. At the bottom layer, the CIDF reconnaissance agents correspond to our data gathering agents. In the middle, the CIDF analysis agents correspond to our low-level agents. At the

top layer, the CIDF decision-response agents correspond to our high-level agents. The CIDF specifies architecture and communication protocol, so it provides a model with which we can compare our system.

The Distributed Intrusion Detection System (DIDS) of the University of California-Davis [Mukherjee94] uses a combination of host monitors and local area network monitors to monitor system & network activities. A centralized director aggregates information from the monitors to detect intrusions. DIDS is similar to our agent system for intrusion detection and countermeasures in that it uses multiple monitors and artificial intelligence algorithms to determine the severity of events. DIDS differs from our system in that the intelligence is purely centralized, and DIDS does not make use of any agent technology.

The Computer Immunology project at the University of New Mexico [Forrest97] explored designs of intrusion detection systems based on animal immune systems. One portion of the project developed a sense of "self" for security-related computer programs by creating a database of normal and abnormal system call traces from instances of execution of the programs [Forrest96]. This sense of self can be used to detect attacks by comparing execution traces of processes to the database of system call traces. Work that is more recent used a variety of approaches to detect intrusions using system call traces from several different privileged programs [Warrender99]. The Computer Immunology project differs from our project by focusing on individual agents rather than feeding the data into a data warehouse.

The Java Agents for Meta-Learning (JAM) project at Columbia University [Stolfo97] is the most similar to our agent system for intrusion detection and countermeasures. JAM uses intelligent, distributed Java agents and data mining to learn models of fraud and intrusive behavior. Our project differs from JAM in that we are concentrating on data mining within an organization. The JAM Project seemed to focus on data mining over multiple organizations. Our project also is looking at countermeasures that could be used to combat intrusions.

The JAM project expanded on the work done by Forrest's group [Forrest96] to detect attacks on privileged programs. Our work attempts to improve on JAM's approach.

The AAFID group at Purdue's COAST project has prototyped an agent-based intrusion detection system. Their paper analyzes the agent-based approach to intrusion detection and mentions the prototype work that has been done on AAFID [Balasubramaniyan98]. Our project differs from AAFID in that we are using data mining to detect attacks on multiple components, emphasizing the use of learning algorithms in intrusion detection, and using mobile agents.

3 Design of Our Agent-Based System

A system of intelligent agents using collaborative information and mobile agent technologies [Bradshaw97] [Nwana96] is developed to implement an intrusion detection system [Denning87].

The goals of the system design are to:

- Learn to detect intrusions on hosts and networks using individual agents targeted at particular subsystems;
- Use mobile agent technologies to intelligently process audit data at the sources;
- Have agents collaborate to share information on suspicious events and determine when to be more vigilant or more relaxed;
- Apply data mining techniques to the heterogeneous data and knowledge sources to identify and react to coordinated attacks on multiple subsystems.

A notable feature of the intrusion detection system based on data mining is the support it offers for gathering and operating on data and knowledge sources from the entire observed system. The system could identify sources of concerted or multistage attacks, initiate countermeasures in response to the attack, and provide supporting documentation for system administrators that would help in procedural or legal action taken against the attacker.

An example of an attack involving more than one subsystem would be a combined NFS and `rlogin` attack. In the first step, an attacker would determine an NFS filehandle for an `.rhosts` file or

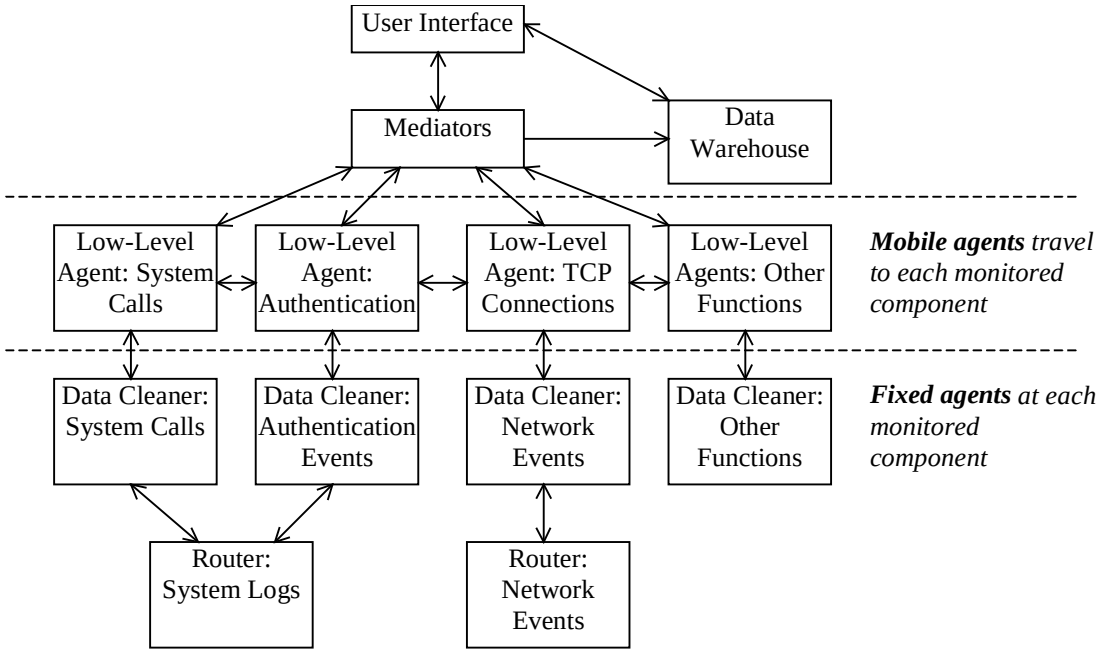

Figure 1: Architecture of the Intrusion Detection System

`/etc/hosts.equiv` (assuming the appropriate filesystems are exported by the UNIX system) [van Doorn94]. Using the NFS filehandle, the attacker would re-write the file to give himself login privileges to the attacked host. Then, using `rlogin` from the formerly untrusted host, the attacker would be able to login to an account on the attacked host, since the attacked host now mistakenly trusts the attacker. At this point, the attacker may be able to further compromise the system. The intrusion detection system based on data mining would be able to correlate these attacks, help identify the origin of the attack, and support system management in responding to the attack.

The components of the agent-based intrusion detection system are shown in Figure 1. Routers read log files and monitor operational aspects of the systems. The routers provide data to the distributed data cleaning agents who have registered their interest in particular data. The data cleaning agents process data obtained from log files, network protocol monitors, and system activity monitors into homogeneous formats. The mobile agents, just above the data cleaning agents in the system architecture, form the first level of intrusion detection. The mobile agents travel to each of their associated data cleaning agents, gather recent information, and classify the data to determine whether suspicious activity is occurring.

Like the JAM system [Stolfo97], the low-level agents may use a variety of classification algorithms. Unlike the JAM system, though, the agents at this level will collaborate to set their suspicion level to determine cooperatively whether a suspicious action is more interesting in the presence of other suspicious activity.

At the top level, mediator agents maintain the data warehouse by combining knowledge and data from the lower layer of agents. The mediator applies data mining algorithms to discover associations and patterns. Because the data warehouse provides a global, temporal view of the knowledge and activity of the monitored distributed system, we believe this system could help train system administrators to spot & defend attacks. We believe our system could also assist system administrators in developing better protections and countermeasures for their systems and identifying new attacks.

The user interface to the agent-based intrusion detection system directs the operation of the agents in the system and shows the status reported by the mobile agents. The user interface also provides access to the data warehouse features.

In our project's current state, several data cleaning and low-level agents have been implemented. This paper discusses the agent that monitors privileged programs using machine learning techniques. Our work in progress includes the integration of data-driven knowledge discovery agents into a distributed knowledge network for monitoring distributed computing systems. In general, we are interested in machine learning approaches to discovering patterns of coordinated attacks on a system wherein individual attacks are spread over space and time.

4 Rule Learning from System Call Traces

Programs that provide network services in distributed computing systems often execute with special privileges. For example, the popular `sendmail` mail transfer agent operates with superuser privileges on UNIX systems. These privileged programs are often a target for intrusions.

The trace of system calls executed by a program can identify whether a program was attacked or misused [Forrest96, Lee98]. We propose that we can compactly represent the system call traces from a particular program by using a feature vector. This compact representation of the process trace can be easily stored in our data warehouse. We further claim that simple rules can be learned by machine learning algorithms. The learned rules are more effective at classifying intrusions than prior techniques.

Forrest's project at the University of New Mexico [Forrest96] developed databases of system calls from normal and anomalous uses of privileged programs such as `sendmail`. Forrest's system call data is a set of files consisting of lines giving a process ID number (PID) and system call number. The files are partitioned based on whether they show behavior of normal or anomalous use of the privileged `sendmail` program running on SunOS 4.1.

Forrest organized system call traces into sequence windows to provide context. Forrest showed that a database of known good sequence windows can be developed from a reasonably sized set of non-intrusive `sendmail` executions. Forrest then showed that intrusive behavior can be determined by finding the percentage of system call sequences that do not match any of the known good sequences. The data sets that were used by Forrest's project are available in electronic form on their Web site [Forrest99]. We believe that our feature vector technique improves on Forrest's technique because it does not depend on a threshold percentage of abnormal sequences. We also find that Forrest's technique does not extract any knowledge from the problem in a way that can be analyzed by experts.

Lee [Lee98] used a portion of the data from Forrest's project to show that the RIPPER [Cohen95] learning algorithm could learn rules from system call sequence windows. Lee organized the system call data into windows of length k+1 with a step size of 1. Values of k=6 and k=10 were found to give the best results in his experiments. For training, each window is assigned a label of "normal" if it matches one of the good windows obtained from proper operations of `sendmail`; otherwise, the window is labeled as "abnormal". An example of the system call windows and labels are shown below in Table 1.

System Call Sequences (k=6)	Label
4, 2, 66, 66, 4, 138, 66	Normal
2, 66, 66, 4, 138, 66, 5	Normal
66, 66, 4, 138, 66, 5, 5	Normal
66, 4, 138, 66, 5, 5, 4	Abnormal
4, 138, 66, 5, 5, 4, 39	Abnormal

Table 1: Sample System Call Windows With Training Labels

After RIPPER is trained, the learned rule set is applied to the testing data to generate classifications for each sequence window. Lee uses a window across the classifications of length 2L+1, where L is the step size for the window, to group labels [Lee98]. If the number of "abnormal" labels in the window exceeds L, the window is considered abnormal. An example of a single window over the classifications is shown below in Table 2, where five classifications are grouped (L=2).

RIPPER's Classification	System Call Sequences (k=6)	Actual Label
Normal	4, 2, 66, 66, 4, 138, 66	Normal
Normal	2, 66, 66, 4, 138, 66, 5	Normal
Abnormal	66, 66, 4, 138, 66, 5, 5	Normal
Abnormal	66, 4, 138, 66, 5, 5, 4	Abnormal
Abnormal	4, 138, 66, 5, 5, 4, 39	Abnormal

Table 2: Sample System Call Windows with Classifications

This scheme filters isolated noise due to occasional prediction errors. When an intrusion takes place, a cluster of system call sequences will usually be classified abnormal. In Table 2, since there are more "abnormal" classifications than "normal" in this window, then this entire window is labeled anomalous. Lee empirically found that values of L=3 and L=5 worked best for identifying intrusions.

Finally, when the window has passed over all the classifications, the percentage of abnormal regions is obtained by dividing the number of anomalous windows by the total number of windows. This percentage is then compared to an empirically derived threshold to determine whether the process trace is an intrusion. We feel that our feature vector technique improves over Lee's technique because it does not depend on an arbitrarily established threshold. We are concerned that establishing an arbitrary threshold is difficult and would require tuning in practice to balance false alarms (false positives) against missed intrusions (false negatives).

5 Representing System Call Traces with Feature Vectors

One of the goals of automated discovery of predictive rules for intrusion detection is to extract the relevant knowledge in a form that lends itself to further analysis by human experts. A natural question that was raised by examination of the rules learned by RIPPER (Cohen, 1995) in the experiments of (Lee and Stolfo, 1998) and (Helmer, Wong, Honavar, and Miller, 1998) was whether essentially the same performance could be achieved by an alternative approach that induced a smaller number of simpler rules. To explore this question, we designed an alternative representation scheme for the training and test data. This representation was inspired by the success of the bag of words representation of documents (Salton, 1983) that has been successfully used by several groups to train text classification systems (Yang, Pai, Honavar and Miller, 1998). In this representation, each document is represented using a vector whose elements correspond to words in the vocabulary. In the simplest case, the vectors are binary and a bit value of 1 indicates that the corresponding word appears in the document in question and bit value of 0 denotes the absence of the word.

In this experiment, the data was encoded as binary-valued bits in feature vectors. Each bit in the vector is used to indicate whether a known system call sequence appeared during the execution of a process. With a sequence window size of 7, 1112 normal sequences and 704 abnormal sequences were seen in the `sendmail` system call data. This encoding is similar in spirit to the bag of words encoding used to represent documents. In our case, the words are system call sequences of length 7. Feature vectors were computed on a per-process basis from the `sendmail` system call traces.

The training data was composed of 80% of the feature vectors randomly selected from normal traces and all of the feature vectors from the arbitrarily selected abnormal traces. The number of abnormal records in the training data was quite small (15 records) in proportion to the set of normal training data (520 records). To balance the weightings, the abnormal training data was duplicated 36 times so that 540 abnormal records were present in the training data. (Lee and Stolfo, 1998) explains the rationale for balancing the data to obtain the desired results from RIPPER. RIPPER efficiently learned a rule set containing 4 simple rules:

```
good IF hf8 = f ay8 = f aa8 = f ak0 = f
good IF ad0 = t
good IF an2 = t an5 = f
```

```
bad otherwise
```

The size of this set of rules compares favorably to the set of 209 rules RIPPER learned when we used Lee's system call window approach.

Applying the learned rule set to the all of the abnormal data and the normal testing data produced the results shown in Table 3. All traces except "Normal sendmail" are intrusions. Boldface traces were used for training. The total number of feature vectors, number of vectors predicted abnormal by RIPPER, and percentage of predicted-abnormal feature vectors is shown. Since each process has only one feature vector, each trace tends to have few feature vectors.

Trace Name	Total Feature Vectors	Feature Vectors Predicted Abnormal	Intrusion Detected?
chasin	6	2	Y
decode1	6	1	Y
decode2	6	1	Y
fwd-loops-1	2	1	Y
fwd-loops-2	1	0	N
fwd-loops-3	2	1	Y
fwd-loops-4	2	1	Y
fwd-loops-5	3	1	Y
recursive	25	3	Y
sm565a	3	1	Y
sm5x	8	2	Y
smdhole	3	2	Y
sscp-1	1	1	Y
sscp-2	1	1	Y
sscp-3	1	1	Y
syslog-local-1	6	6	Y
syslog-local-2	6	5	Y
syslog-remote-1	7	7	Y
syslog-remote-2	4	4	Y
Normal sendmail	120	1	

Table 3: Results of Learning Rules for Feature Vectors

The rules can not be expected to identify all of the processes in a trace from an intrusion because some of the processes involved in handling an intrusive transaction may be indistinguishable from processes handling a normal transaction. If at least one of the processes involved in an intrusion is flagged as abnormal, we can identify that an intrusion has taken place. Intrusions are clearly identified in our experiment with the exception of one of the minor intrusions, fwd-loops-2.

A benefit of the feature vector approach is the simplicity of the learned rules. Training takes place "off line" due to the amount of time need to learn a rule set. Each learned rule set for the `sendmail` system call feature vectors is simple: generally fewer than 10 rules, where each rule often consists of a conjunction of one or two Boolean terms. Such a small set of rules applied to this simple data structure should allow us to use this approach in a near real-time intrusion detection agent without placing an excessive load on a system. A small, simple rule set also lends itself to expert examination and analysis.

6 Feature Subset Selection Using Genetic Algorithms

A learning algorithm's performance in terms of learning time, classification accuracy on test data, and comprehensibility of the learned rules often depends on the features or attributes used to represent the examples. Feature subset selection has been shown to improve the performance of a learning algorithm

and reduce the effort and amount of data required for machine learning on a broad range of problems (Motoda and Liu, 1998). A discussion of alternative approaches to feature subset selection can be found in (John, Kohavi, and Pfleger 1994; Yang and Honavar 1998b; Motoda and Liu, 1998).

The benefits and affects of feature subset selection include:

- Feature subset selection affects the accuracy of a learning algorithm because the features of a data set represent a language. If the language is not expressive enough, the accuracy of any learning algorithm is adversely affected.
- Feature subset selection reduces the computational effort required by a learning algorithm. The size of the search space depends on the features; reducing the feature set to exclude irrelevant features reduces the size of the search space and thus reduces the learning effort.
- The number of examples required to learn a classification function depends on the number of features (Langley 1995, Mitchell 1997). More features require more examples to learn a classification function to a desired accuracy.
- Feature subset selection can also result in lower cost of classification (because of the cost of obtaining feature values through measurement or simply the computation overhead of processing the features).

Against this background, it is natural to consider feature subset selection as a possible means of improving the performance of machine learning algorithms for intrusion detection (Frank 1994).

Genetic algorithms and related approaches [Goldberg89, Michalewicz96, Koza92] offer an attractive alternative to exhaustive search (which is infeasible in most cases due to its computational complexity). They also have an advantage over commonly used heuristic search algorithms that rely on the monotonicity assumption (i.e., addition of features does not worsen classification accuracy) which is often violated in practice (Yang and Honavar, 1998b).

The genetic algorithm for feature subset selection starts with a randomly generated population of individuals, where each individual corresponds to a candidate feature subset. Each individual is encoded as a string of 0's and 1's. The number of bits in the string is equal to the total number of features. A 1 in the bit string indicates an attribute is to be used for training, and a 0 indicates that the attribute should not be used for training. The fitness of a feature subset is measured by the test accuracy (or cross-validation accuracy of the classifier learned using the feature subset) and any other criteria of interest (e.g., number of features used, the complexity of the rules learned). In our case, we used the RIPPER rule learning algorithm to train the classifier. It is set up to treat any attributes that are not selected as missing. The training data is provided to RIPPER's `model()` function, which learns a rule set from the data. The number of conditions in the learned rule set is counted by RIPPER's `concept_size()` function, and this value is used to determine the complexity of the learned hypothesis. The learned rule set is applied to the test examples and the determined accuracy is returned to the feature subset selection routine. The fitness of the individual is calculated, based on the number (n) of attributes used in learning, the accuracy of the learned hypothesis (a) and the total number of attributes (N):

$$a - \frac{n}{a} + N$$

This fitness is then used to rank the individuals for selection.

A primary goal in using feature subset selection on this intrusion detection problem is to improve accuracy. A high percentage of the intrusion detection alerts reported by current intrusion detection systems are false alarms. Our system needs to be highly reliable, and we would like to keep false alarms to a minimum. A secondary goal is to reduce the amount of data that must be obtained from running processes and classified. This would reduce the overhead of our intrusion detection approach on the monitored system.

7 Feature Subset Selection Results

The genetic algorithm used standard mutation and crossover operators with 0.001 probability of mutation and 0.6 probability of crossover with rank-based selection (Goldberg, 1989). The probability of selecting the best individual was 0.6. A population size of 50 was used and each run went through 5 generations.

There are 1816 Boolean-valued features in each `sendmail` system call feature vector. The data used by the genetic algorithm was the same as the training data used for the previous feature vector experiment (1060 feature vectors), with the addition of an additional copy of each unique feature vector in the training data (72 feature vectors). This was done to ensure that rare but potentially important cases had a reasonable probability of being sampled in the training and testing phases. This gave a total of 1132 feature vectors in the input to the genetic algorithm.

To show the general effectiveness of genetic feature selection on this problem, Table 4 shows the results of five separate runs of the genetic algorithm with RIPPER with identical parameters used for each run. The number of attributes is significantly reduced while the accuracy is maintained.

Trial	Training Accuracy of Best Individual	Attributes Used in Best Individual
1	98.9399	847
2	98.8516	857
3	99.1166	846
4	99.1166	849
5	99.1166	839

Table 4: Feature Subset Selection Results with Constant Parameters

Table 5 shows the results of using the rules from the best individuals found in the five genetic feature selection runs and compares the results to the original results learned from all the features. All traces except “Normal sendmail” are intrusions. Boldface traces were used for training. Despite using only about half the features in the original data set, the performance of the learned rules was comparable to that obtained using the entire set of features. After feature subset selection, none of the feature vectors from normal sendmail are labeled as abnormal. This shows an improvement in the rate of false positives.

Trace Name	Attack Detected: All Attributes	Attack Detected: Feature Selection Trial 1	Attack Detected: Feature Selection Trial 2	Attack Detected: Feature Selection Trial 3	Attack Detected: Feature Selection Trial 4	Attack Detected: Feature Selection Trial 5
chasin	Y	Y	Y	Y	Y	Y
decode1	Y	Y	Y	Y	Y	Y
decode2	Y	Y	Y	Y	Y	Y
fwd-loops-1	Y	Y	Y	Y	Y	Y
fwd-loops-2	N	N	Y	N	N	N
fwd-loops-3	Y	Y	Y	Y	Y	Y
fwd-loops-4	Y	Y	Y	Y	Y	Y
fwd-loops-5	Y	Y	Y	Y	Y	Y
recursive	Y	Y	Y	Y	Y	Y
sm565a	Y	Y	Y	Y	Y	Y
sm5x	Y	Y	Y	Y	Y	Y
smdhole	Y	Y	Y	Y	Y	Y
sscp-1	Y	Y	Y	Y	Y	Y
sscp-2	Y	Y	Y	Y	Y	Y
sscp-3	Y	Y	Y	Y	Y	Y
syslog-l1	Y	Y	Y	Y	Y	Y
syslog-l2	Y	Y	Y	Y	Y	Y

syslog-r1	Y	Y	Y	Y	Y	Y
syslog-r2	Y	Y	Y	Y	Y	Y
Normal sendmail	1/120	0/120	0/120	0/120	0/120	0/120

Table 5: Results from Rules Learned by Genetic Feature Selection

7.1 Analysis of Learning Algorithms

A summary of the effectiveness of RIPPER and RIPPER with the genetic feature selection algorithm follows.

Measure	RIPPER on System Call Windows	RIPPER on Feature Vectors	RIPPER with Genetic Feature Selection on Feature Vectors
Learning Effort	Moderate (30 minutes)	Very Good (under 1 minute)	Intensive (approx. 4 hours)
Accuracy of Learned Hypothesis	Good (0.53% False Abnormal)	Good (0.83% False Abnormal)	Very Good (0% False Abnormal)
Complexity of Learned Hypothesis	Poor (Avg. 225 rules)	Good (4 rules, 7 tests)	Good (Avg. 8.6 rules, 9.6 tests)
Number of Attributes Used	7 (7 system calls in window)	1816	Avg. 848.9
Classification Effort	Moderate (large rule set)	Small (trivial rule set)	Smaller (trivial rule set, fewer features)

Table 6: Effectiveness of Different Learning Techniques

Table 6 illustrates the advantages of the feature vector representation over the system call windows for this learning problem. The feature vector representation allows the learning algorithm to learn a hypothesis much faster and with comparable accuracy on the normal test data, and the complexity of the hypothesis is much smaller. Using genetic feature selection on the feature vectors is time consuming but further improves the learned hypothesis and reduces the set of attributes used for learning.

7.2 Analysis of Rules Learned by RIPPER

An example set of rules that were learned in first trial of RIPPER with genetic feature subset selection is shown below:

```
good IF abnk = t .
good IF aabb = t .
good IF abhp = f AND aafk = f .
good IF aaam = t .
good IF aahj = t .
good IF aaip = t .
good IF aamp = t .
bad IF .
```

The set above contains 8 individual rules composed of 8 tests, which correspond to this pseudocode:

IF “unlink,close,unlink,unlink,close,gettimeofday,open” seen THEN good
ELSE IF “chmod,ioctl,fstat,write,close,unlink,rename” seen THEN good
ELSE IF “sigsetmask,sigblock,sigvec,sigvec,sigsetmask,sigblock,sigvec” not seen and
“close,setitimer,close,gettimeofday,link,socket,fcntl” not seen THEN good
ELSE IF “accept,wait4,wait4,wait4,wait4,accept,fork” seen THEN good
ELSE IF “fcntl,gettimeofday,getpid,sendto,accept,fork,close” seen THEN good
ELSE IF “fstat,mmap,close,open,fstat,mmap,getdents” seen THEN good
ELSE IF “getpid,sendto,accept,wait4,wait4,accept,close” seen THEN good
ELSE bad

Each of the rule sets from the five genetic algorithm trials contains rules that can be found in the other rule sets. The third and fourth trials contain mostly unique rules, while the other three runs contain a majority of rules that are duplicated in other rule sets. Because the rule sets identify normal processes and consider all others abnormal, none of the rules looks for the existence of abnormal system call sequences. Consequently, we did not notice any particular system call sequences that would directly identify an intrusion.

In general, the small size of the rules sets learned by RIPPER from the system call feature vectors and the performance of these learned rule sets indicates that a concise set of rules clearly distinguish good `sendmail` processes from those that have been attacked.

8 Conclusion and Future Work

Intrusion detection and abuse detection in computer systems in networked environments is a problem of great practical interest. This paper investigated the classification of system call traces for intrusion detection. From a feature vector representation of system call traces RIPPER learned a small, concise set of rules that was successful at classifying intrusions. It was further shown that feature subset selection reduced the number of features in the data, which resulted in less data and effort required for training due to the smaller search space. Feature selection also gave equivalent accuracy with a smaller set of features.

We have integrated the learned rules into a mobile agent running on a distributed system consisting of Pentium II systems running FreeBSD. This laboratory network is connected by a firewall to the Department of Computer Science’s network so we may operate the intrusion detection system in a controlled environment. During operation of the IDS, a Voyager server is started on each host in the monitored distributed system, and the mobile agent is able to travel through the system, classify sample sendmail system call feature vectors, and report the results to the user interface. We have implemented a set of Java classes that can interpret and apply the RIPPER rules, which allows our mobile agent to move the classifier with it as it travels through the distributed system.

The rule sets learned using the feature vector representation are an order of magnitude simpler than those obtained using other approaches reported in the literature (Helmer, Wong, Honavar, and Miller, 1998) and (Lee and Stolfo, 1998). This is especially noteworthy given the fact that all of the experiments in question used the same rule learning algorithm. We conjecture that the feature vector representation used in our experiments is primarily responsible for the differences in the rule sets that are learned.

Open issues include the use of this technique in heterogeneous distributed systems. System call traces for a particular privileged program should not differ greatly among various UNIX operating system implementations. We hope that the similarities would be strong enough to allow for a single rule set that would apply to several different systems.

Another issue is whether this technique could be applied in real time. Feature subset selection itself is computationally expensive, so training and refining the agent can not be done in real time. After the agent is trained, our technique can determine whether a process is an intruder only after the process has finished, which provides near real time detection. Either Warrender’s or Lee’s techniques [Warrender98, Lee98] would allow anomaly detection in real time during the execution of the process. We believe that our technique could be refined to determine the likelihood that a process is intrusive during the process’

execution, giving real time detection. This refinement would be necessary for long-lived daemons such as HTTP servers.

We would also like to know how well this technique applies to privileged programs other than sendmail. Warrender worked with five distinct privileged programs and identified cases where different thresholds and/or different algorithms worked better for different programs. Based on her work, we expect this technique will be successful for more programs than just `sendmail`.

Work in progress on intrusion detection is aimed at the integration of data-driven knowledge discovery agents into a distributed knowledge network for monitoring and protection of distributed computing systems and information infrastructures. The investigation of machine learning approaches to discover patterns of coordinated attacks on a system wherein individual attacks are spread over space and time is of particular interest in this context.

Acknowledgements

This work was supported by the Department of Defense under contract MDA 904-98-C-A887.

Thanks go to the Computer Immune System Project at the University of New Mexico's Computer Science Department for the use of their `sendmail` system call data.

Bibliography

[Balasubramaniyan98] Balasubramaniyan, Jai, Jose Omar Garcia-Fernandez, E. H. Spafford, and Diego Zamboni. An Architecture for Intrusion Detection using Autonomous Agents. Department of Computer Sciences, Purdue University; Coast TR 98-05; 1998.

[Bradshaw97] Bradshaw, J.M. An Introduction to Software Agents. In: Bradshaw, J.M. (ed.), *Software Agents*, Cambridge, MA: MIT Press, 1997.

[Cohen95] Cohen, W. W. Fast Effective Rule Induction. Proceedings of the 12th International Conference on Machine Learning, Lake Tahoe, CA, 1995. Morgan Kaufmann.

[Crosbie95] Crosbie, M. and G. Spafford. Defending a Computer System using Autonomous Agents. In Proceedings of the 18th National Information Systems Security Conference, October 1995.

[Denning87] Denning, Dorothy. An Intrusion-Detection Model. IEEE Transactions on Software Engineering, Number 2, page 222, February 1987.

[Forrest99] Forrest, Stephanie. Computer Immune Systems Research. Available online at http://www.cs.unm.edu/~immsec/. 1999.

[Forrest97] Forrest, S., S. Hofmeyr, and A. Somayaji. Computer Immunology. Communications of the ACM, 40(10):88-96, November 1997.

[Forrest96] Forrest, S., S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A Sense of Self for UNIX Processes. Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages 120-128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[Frank94] Frank, Jeremy. Artificial Intelligence and Intrusion Detection: Current and Future Directions. Proceedings of the 17th National Computer Security Conference, 1994.

[Goldberg89] Goldberg, D. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, New York. 1989.

[Helmer98] Helmer, Guy, Johnny S. K. Wong, Vasant Honavar, and Les Miller. Intelligent Agents for Intrusion Detection and Countermeasures. Proceedings, 1998 IEEE Information Technology Conference, Syracuse, NY. 1998.

[John94] John, G., R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. Proceedings of the Eleventh International Conference on Machine Learning, pp. 121-129, New Brunswick, NJ. Morgan Kaufmann. 1994.

[Koza92] Koza., J. *Genetic Programming*. Boston, MA: MIT Press. 1992.

[Lee98] Lee, W. and S. Stolfo. Data Mining Approaches for Intrusion Detection. Proceedings, 7th USENIX Security Symposium. 1998.

[Michalewicz96] Michalewicz, M. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, New York. 1996. Third Edition.

[Mitchell96] Mitchell, M. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA. 1996.

Motoda and Liu, eds. Feature Extraction, Construction, and Selection—A Data Mining Perspective. Kluwer Academic Publishers, Boston. 1998.

[Mukherjee94] Mukherjee, Biswanath, L. Todd Heberlein, and Karl N. Levitt. Network Intrusion Detection. IEEE Network, 8(3):26-41, May/June 1994.

[ObjectSpace97] ObjectSpace, Inc. ObjectSpace Voyager Core Technology User Guide. Available online at `http://www.objectspace.com/voyager/whitepapers/Voyager.PDF`. 1997. Version 1.0.0.

[Porras98] Porras, Phillip A. and Alfonso Valdes. Live Traffic Analysis of TCP/IP Gateways. Proceedings, Networks and Distributed Systems Security Symposium, March 1998.

[Reilly98] Reilly, Mark and Maureen Stillman. Open Infrastructure for Scalable Intrusion Detection. Proceedings, 1998 IEEE Information Technology Conference, Syracuse, NY. 1998.

[Salton83] Salton, G. *Automated Text Processing*. New York: Prentice Hall. 1983.

[Stolfo97] Stolfo, S., A. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P. Chan. JAM: Java Agents for Meta-learning over Distributed Databases. AAAI97 Workshop on AI Methods in Fraud and Risk Management.

[van Doorn94] van Doorn, Leendert. `nfsbug.c`. Available online at `http://www.asmodeus.com/archive/Xnix/nfsbug/nfsbug.c`. 1994.

[Yang98a] Yang, Jihoon, and Vasant Honavar. Feature Subset Selection Using a Genetic Algorithm. IEEE Intelligent Systems Special Issue: Feature Transformation and Subset Selection. 1998.

[Yang98b] Yang, Jihoon, and Vasant Honavar. Feature Subset Selection Using a Genetic Algorithm. Feature Extraction, Construction, and Selection—A Data Mining Perspective, Liu and Motoda, eds., Kluwer Academic Publishers, Boston. 1998.

[Warrender99] Warrender, Christina, Stephanie Forrest, and Barak Pearlmutter. Detecting Intrusions Using System Calls: Alternative Data Models. To appear, 1999 IEEE Symposium on Security and Privacy. 1999.