ELSEVIER

The Journal of Systems and Software 60 (2002) 165–175

The Journal of Systems and Software

www.elsevier.com/locate/jss

Automated discovery of concise predictive rules for intrusion detection

Guy Helmer, Johnny S.K. Wong *, Vasant Honavar, Les Miller

Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, IA 50011-1041, USA

Received 12 July 2000; accepted 21 February 2001

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 stationary 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 attack. 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 improving the accuracy of predictions.

Keywords: Intrusion detection; Machine learning; Feature subset selection

1. Introduction

A definition of an intrusion is "any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource" (Heady et al., 1990). Detecting intrusions against distributed computing systems is a difficult problem (Denning, 1987). 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 et al., 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 intrusions spread over space and time) and the initiation of suitable countermeasures.

Our current implementation includes:

- Mobile and stationary 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;
- 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 intrusions over the whole system, execute countermeasures, and support the system administration in their pursuit of attackers.

* Corresponding author. Tel.: +1-515-294-2586; fax: +1-515-294-0258.

E-mail addresses: ghelmer@cs.iastate.edu (G. Helmer), wong@cs.iastate.edu (J.S.K. Wong), honavar@cs.iastate.edu (V. Honavar), lmiller@cs.iastate.edu (L. Miller).

0164-1212/02/$ - see front matter
PII: S0164-1212(01)00088-7

The data warehouse knowledge provided by system of agents could be used to help systems management staff learn misuse patterns, understand intrusions, and support offensive and defensive actions. Our intrusion detection system has been implemented using Java and ObjectSpace's Voyager mobile agent facility (ObjectSpace Inc, 1999).

In this paper, we sketch the design of our distributed intrusion detection system to provide background that explains how the system call intrusion detector fits into our larger system. We then explain our research in the application of artificial intelligence to the problem of identifying misuse of privileged programs. The Computer Immunology project (Forrest et al., 1996) and the Java Agents for Meta-Learning project (Lee and Stolfo, 1998) 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. We show that our feature vector representation works well for automated knowledge discovery using rule learning. We then employ feature subset selection using a genetic algorithm to reduce the number of features necessary in the feature vector and to improve the accuracy of the results by eliminating extraneous features.

2. Related work

Projects closely related to our intrusion detection project include Common Intrusion Detection Framework (CIDF), Distributed Intrusion Detection System (DIDS), Computer Immunology, Java Agents for Meta-Learning (JAM) and AAFID.

The CIDF (Reilly and Stillman, 1998) 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 similarity of our prototype system to the CIDF model and the flexibility of the CISL language (Feiertag et al., 1999) should lend itself to integration with other CIDF-compatible systems when the need arises.

The DIDS of the University of California-Davis (Mukherjee et al., 1994) uses a combination of host monitors and local area network monitors to monitor system and 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 (Forrest et al., 1997) 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 system call traces from instances of execution of the programs (Forrest et al., 1996). This sense of self can be used to detect intrusions 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 (Warrender et al., 1999).

In general, the Computer Immunology project differs from our project by focusing on individual agents rather than sharing data between agents and feeding the data into a data warehouse. In our specific application of intrusion detection using system calls, we do not attempt to develop a sense of self but instead concentrate on developing rules to check a particular program's execution trace for anomalies. We assume knowledge of the identify of the program being traced and apply a rule set learned for that particular program.

The JAM project at Columbia University (Stolfo et al., 1997) 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. The JAM project expanded on the work done by Forrest's group (Forrest et al., 1996) to detect intrusions on privileged programs. Like the JAM project, we apply the data mining approach to intrusion detection but use a different data representation that provides a single signature for each process.

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 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 (Balasubramaniyan et al., 1998). Our project differs from AAFID in that we are using data mining to detect intrusions on multiple components, emphasizing the use of learning algorithms in intrusion detection, and using mobile agents. AAFID is implemented in Perl while our system is implemented in Java.

3. Design of our agent-based system

A system of intelligent agents using collaborative information and mobile agent technologies (Bradshaw, 1997; Nwana, 1996) is developed to implement an intrusion detection system (Denning, 1987).

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 intrusions 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 intrusions, initiate countermeasures in response to the intrusion, and provide supporting documentation for system administrators that would help in procedural or legal action taken against the attacker.

An example of an intrusion involving more than one subsystem would be a combined NFS and rlogin intrusion. In the first step, an attacker would determine an NFS filehandle for an .rhosts file or /etc/hosts.equiv (assuming the appropriate filesystems are exported by the UNIX system) (van Doorn, 1999). 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 intrusions, help to identify the origin of the intrusion, and support system management in responding to the intrusion. The components of the agent-based intrusion detection system are shown in Fig. 1. Information routers read log files and monitor operational aspects of the systems. The information 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 (Stolfo et al., 1997), 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.

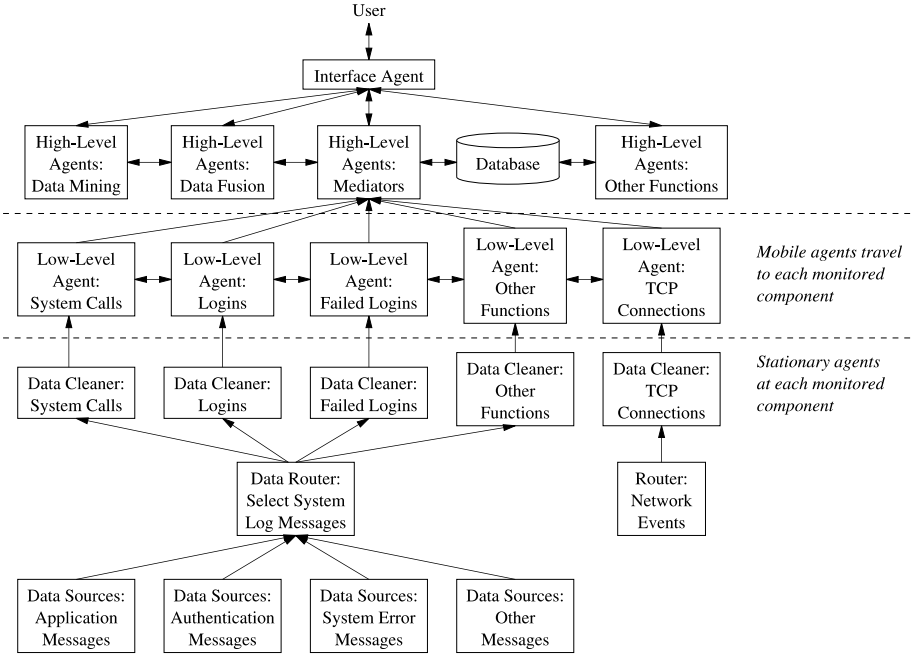

Fig. 1. Architecture of the intrusion detection system.

At the top level, high-level agents maintain the data warehouse by combining knowledge and data from the low-level agents. The high-level agents apply 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, this system could help train system administrators to spot and defend intrusions. Our system could also assist system administrators in developing better protections and countermeasures for their systems and identifying new intrusions.

The interface agent for the agent-based intrusion detection system directs the operation of the agents in the system and maintains the status reported by the mobile agents. The interface agent 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 intrusions on a system wherein individual intrusions 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. Privileged programs like sendmail are often a target for intrusions.

The trace of system calls executed by a program can identify whether an intrusion was mounted against a program (Forrest et al., 1996; Lee and Stolfo, 1998). Forrest's project at the University of New Mexico (Forrest et al., 1996) 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 (Forrest, 1999). We use the same data set to enable comparison with techniques used in related papers (Lee and Stolfo, 1998; Warrender et al., 1999).

Our feature vector technique improves on Forrest's technique because it does not depend on a threshold percentage of abnormal sequences. Our feature vector technique compactly summarizes the vast data obtained from each process, enabling longer-term storage of the data for reference and analysis. With respect to other rule learning techniques, our technique induces a compact rule set that is easily carried in lightweight agents. Our technique also may mine knowledge from the data in a way that can be analyzed by experts.

Lee and Stolfo (1998) used a portion of the data from Forrest's project to show that the RIPPER (Cohen, 1995) learning algorithm could learn rules from system call sequence windows. Lee empirically found sequences of length 7 and 11 gave the best results in his experiments (Lee and Stolfo, 1998). 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 in Table 1. 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 (Lee and Stolfo, 1998). 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 in Table 2.

The window scheme filters isolated noise due to occasional prediction errors. When an intrusion takes

Table 1
Sample system call windows with training labels

System call sequences	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 2
Sample system call windows with classifications

RIPPER's classification	System call sequences	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

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 (Lee and Stolfo, 1998).

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. Lee uses this percentage to empirically derive a threshold that separates normal processes from anomalous processes. Warrender et al. (1999) uses a similar technique, the Locality Frame Count (LFC), that counts the number of mismatches in a group and considers the group anomalous if the count exceeds a threshold. Warrender's technique allows intrusion detection for long-running daemons, where an intrusion could be masked by a large number of normal windows with Lee's technique.

Lee and Stolfo (1998) developed an alternate technique that predicts one of the system calls in a sequence. The alternate technique allows learning of normal behavior in the absence of anomalous data. Our technique is less suitable in that is does require anomalous data for training.

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 et al. (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 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 et al., 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 were 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. This encoding is similar in spirit to the bag of words encoding used to represent text documents.

Feature vectors were computed on a per-process basis from the sendmail system call traces (Forrest, 1999). Based on ideas from previous work (Forrest et al., 1996; Lee and Stolfo, 1998), sequence windows of size 5–12 were evaluated for use with our feature vector approach. Sequence windows of size 7 were selected for their good performance in learning accuracy and relatively small dictionary size.

The training data was composed of 80% of the feature vectors randomly selected from normal traces and all of the feature vectors from the selected abnormal traces. To compare our results to those from the JAM project, four specific anomalous traces were selected for training. Five different selections of anomalous traces were also tested to ensure that arbitrarily selecting these four anomalous traces did not significantly affect the results.

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. From the feature vectors built from sequences of length 7, RIPPER efficiently learned a rule set containing seven simple rules:

```
good IF a1406 = t
good if a67 = t
good if a65 = t
good if a576 = t
good if a132 = t
good if a1608 = t
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. The feature vector approach condenses information about an entire process' history of execution. Feature vectors may make it easier for learning algorithms by aggregating information over the entire execution of a process rather than by looking at individual sequences.

Applying the learned rule set produced the results shown in Table 3. All traces except "Normal sendmail" are anomalous. Boldface traces were used for training. The total numbers of feature vectors, numbers of vectors predicted abnormal by RIPPER, and detection results are shown. Since a single feature vector represents each process, each trace tends to have few feature vectors.

The rules cannot be expected to flag all of the processes in an attacked trace as an intrusion. While handling a mail message, sendmail spawns child processes that handle different parts of the procedures involved in receiving, queuing, and forwarding or delivering the message. Some of these processes involved in handling

Table 3
Results of learning rules for feature vectors

Trace name	Total feature vectors	Vectors predicted abnormal	Attack detected?
chasin	6	3	Y
decode1	6	2	Y
decode2	6	2	Y
fwd-loops-1	2	2	Y
fwd-loops-2	1	0	N
fwd-loops-3	2	2	Y
fwd-loops-4	2	2	Y
fwd-loops-5	3	2	Y
recursive	25	23	Y
sm565a	3	2	Y
sm5x	8	3	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	6	Y
syslog-remote-1	7	7	Y
syslog-remote-2	4	4	Y
Normal sendmail (not used for training)	130	3	

an intrusive transaction may be indistinguishable from processes handling a normal transaction because the attack only affects one of the processes. Therefore, if at least one of the processes involved in an intrusion is flagged as abnormal, we can identify the group of related processes as anomalous.

Several attacks did not result in successful intrusions. For our intrusion detection system, we identify all attacks as intrusive activity that merits further investigation elsewhere in the IDS. It would be unlikely that an attacker would attempt a single exploit and give up if it fails. The data mining portion of our intrusion detection system would then correlate these multiple (successful and unsuccessful) attacks.

The anomalous traces are clearly identified in our experiment with the exception of one of the minor intrusions, fwd-loops-2. The fwd-loop attacks are denial-of-service attacks where the sendmail process spends its time repeatedly forwarding the same message. The feature vector technique may need to be adjusted from simple binary values to statistical measures to identify this class of attack.

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 may lend itself to human expert examination and analysis in data mining situations (Cabena et al., 1998).

Another benefit of the feature vector approach is the condensed representation of a process by its fixed-length feature vector. The list of system calls executed by a process can be enormous. Storing this information in its entirety is infeasible. Representing the data by a relatively short fixed-length string helps solve the problems of transmitting and storing the data. This technique realizes the mobile agent architecture's goal of reducing and summarizing data at the point of generation.

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 (Liu and Motoda, 1998). A discussion of alternative approaches to feature subset selection can be found in John et al. (1994), Yang and Honavar (1998), Liu and Motoda (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 (Goldberg, 1989; Michalewicz, 1996; Koza, 1992) 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, 1998).

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).

We used the RIPPER rule learning algorithm as the classifier. The training data is provided to RIPPER, which learns a rule set from the data. The number of conditions in the learned rule set is counted, 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 accuracy of the learned hypothesis ($accuracy(x)$), the number of attributes ($\text{cost}(x)$) used in learning, the complexity of the learned hypothesis ($\text{complexity}(x)$), and weights (w_{accuracy}, w_{cost}, $w_{\text{complexity}}$) for each parameter:

$$\text{fitness}(x) = w_{\text{accuracy}} * \text{accuracy}(x) + w_{\text{cost}} * \text{cost}(x) + w_{\text{complexity}} * \text{complexity}(x).$$

This fitness is then used to rank the individuals for selection. Other methods of computing fitness are possible and are discussed by Yang and Honavar (1998).

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.

6.1. 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 five generations.

Table 4
Feature subset selection results with constant parameters

Trial	Training accuracy of best individual	Attributes used by best individual
1	98.9399	847
2	98.8516	857
3	99.1166	846
4	99.1166	849
5	99.1166	839

We started with the training data used for the previous feature vector experiment (1060 feature vectors). We added an additional copy of each unique feature vector in the training data (72 feature vectors) 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.

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.

7. Analysis

A comparison of the effectiveness of RIPPER on the problem using two different data representations and genetic feature selection algorithm follows.

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.

Table 5
Results from rules learned by genetic feature selection

Trace	All attributes	Trial 1	Trial 2	Trial 3	Trial 4	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-local-1	Y	Y	Y	Y	Y	Y
syslog-local-2	Y	Y	Y	Y	Y	Y
syslog-remote-1	Y	Y	Y	Y	Y	Y
syslog-remote-2	Y	Y	Y	Y	Y	Y
Normal sendmail	1/120	0/120	0/120	0/120	0/120	0/120

Table 6
Effectiveness of different learning techniques

Measure	Sequence windows	Feature vectors	Genetic algorithm feature selection
Learning effort	Moderate (30 min)	Very good (under 1 min)	Intensive (approx. 4 h)
Accuracy of learned hypothesis	Good (0.53% false positive)	Good (0.83% false positive)	Very good (0% false positive)
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)	1832	Avg. 848.9
Classification effort	Moderate (large rule set)	Small (trivial rule set)	Smaller (trivial rule set, fewer features)

7.1. 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 a1024 = t.
good IF a27 = t.
good IF a873 = f AND a130 = f.
good IF a12 = t.
good IF a191 = t.
good IF a223 = t.
good IF a327 = t.
bad IF.
```

The set above contains eight individual rules composed of eight 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
ELSEIF "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. The similarities of rules between runs likely indicates the strength of particular sequences in identifying normal behavior.

Because the rule sets identify normal processes and consider all others abnormal, none of the rules identifies particular abnormal system call sequences. Consequently, the rules do not identify system call sequences that would directly signal an intrusion. However, these rules may lead to an understanding of how an attack causes the typical sequence of system calls to change.

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 normal sendmail processes from anomalous.

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 through the technique of describing each process by a feature vector. From the feature vector representation RIPPER learned a small, concise set of rules that was successful at classifying intrusions. In comparison with other techniques, the feature vector representation does not depend on thresholds to separate normal from anomalous. 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).

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 et al., 1998; 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. The feature vectors condense information from the entire execution of a process compared to the fine-grained detail of individual sequences. The scope of information contained in the feature vectors may make it easier for learning algorithms to learn simple rules.

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. For operation of the IDS, a Voyager server is started on each host in the monitored distributed system. The mobile agent is travel through the system, classifies sample sendmail system call feature vectors, and reports the results to its mediator. The mediator reports the results to the user interface and optionally stores the information in a database for potential mining and warehousing operations. We have implemented a set of Java classes that can interpret and apply the RIPPER rules, which allows our mobile agent to bring its classifier and rule set(s) with it as it travels through the distributed system.

Open issues include the use of this technique in heterogeneous distributed systems. Specific rule sets may need to be developed for each node in a distributed system due to variabilities between operating systems and workload characteristics. Fortunately, the rule sets discovered by RIPPER have been small, so mobile agents ought to be able to carry multiple rule sets without becoming overly "heavy".

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 cannot 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. Warrender et al. (1999) or Lee and Stolfo (1998) techniques would allow anomaly detection in real time during the execution of the process. 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 (Warrender et al., 1999). 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 intrusions on a system wherein individual intrusions are spread over space and time is of particular interest in this context.

Acknowledgements

This work was supported by the Department of Defense. Thanks 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.

References

Balasubramaniyan, J., Garcia-Fernandez, J.O., Isacoff, D., Spafford, E.H., Zamboni, D., 1998. An architecture for intrusion detection using autonomous agents. Technical Report COAST TR 98-05, Purdue University Department of Computer Sciences.

Bradshaw, J.M. (Ed.), 1997. An Introduction to Software Agents. MIT Press, Cambridge, MA.

Cabena, P., Hadjinian, P., Stadler, R., Verhees, J., Zanasi, A., 1998. Discovering Data Mining: From Concept to Implementation. Prentice-Hall, Upper Saddle River, NJ.

Cohen, W.W., 1995. Fast effective rule induction. In: Proceedings of the 12th International Conference on Machine Learning, Lake Tahoe, CA. Morgan Kaufmann, Los Altos, CA.

Denning, D.E., 1987. An intrusion-detection model. IEEE Trans. Software Eng., SE 13 (2), 222–232.

Feiertag, R., Kahn, C., Porras, P., Schnackenberg, D., Staniford-Chen, S., Tung, B., 1999. A common intrusion specification language (CISL).. Available from http://www.isi.edu/gost/projects/crisis/cidf/cisl_current.txt.

Forrest, S., 1999. Computer immune systems data sets.. Available from http://www.cs.unm.edu/~immsec/data-sets.htm.

Forrest, S., Hofmeyr, S.A., Somayaji, A., 1997. Computer immunology. Commun. ACM 40 (10), 88–96.

Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A., 1996. A sense of self for Unix processes. In: Proceedings of the 1996 IEEE Symposium on Security and Privacy, Los Altos, CA. IEEE Computer Society Press, Silver Spring, MD, pp. 120–128.

Frank, J., 1994. Artificial intelligence and intrusion detection: Current and future direction. In: Proceedings of the 17th National Computer Security Conference.

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

Heady, R., Luger, G., Maccabe, A., Servilla, M., 1990. The architecture of a network level intrusion detection system. Technical Report CS90-20, Department of Computer Science, University of New Mexico, August.

Helmer, G., Wong, J.S.K., Honavar, V., Miller, L., 1998. Intelligent agents for intrusion detection. In: Proceedings IEEE Information Technology Conference, Syracuse, NY, pp. 121–124.

Honavar, V., Miller, L., Wong, J.S.K., 1998. Distributed knowledge networks. In: Proceedings IEEE Information Technology Conference, Syracuse, NY, pp. 87–90.

John, G., Kohavi, R., Pfleger, K., 1994. Irrelevant features and the subset selection problem. In: Proceedings of the 11th International Conference on Machine Learning, New Brunswick, NY. Morgan Kaufmann, Los Altos, CA, pp. 121–129.

Koza, J., 1992. Genetic Programming. MIT Press, Boston, MA.

Langley, P., 1995. Elements of Machine Learning. Morgan Kaufmann, Palo Alto, CA.

Lee, W., Stolfo, S., 1998. Data mining approaches for intrusion detection. In: Proceeding of the Seventh USENIX Security Symposium, USENIX.

Liu, H., Motoda, H. (Eds.), 1998. Feature Extraction, Construction, and Selection – A Data Mining Perspective. Kluwer Academic Publishers, Boston, MA.

Michalewicz, M., 1996. Genetic Programs Algorithms + Data Structures = Evolution, third ed. Springer, New York.

Mitchell, T.M., 1997. Machine Learning. McGraw Hill, New York.

Mukherjee, B., Heberlein, L.T., Levitt, K.N., 1994. Network intrusion detection. IEEE Network 8 (3), 26–41.

Nwana, H.S., 1996. Software agents: An overview. Knowledge Eng. Rev. 11 (3), 205–244.

ObjectSpace Inc, Dallas, TX. ObjectSpace Voyager Core Technology User Guide, 1999. ver. 3.0.0.

Reilly, M., Stillman, M., 1998. Open infrastructure for scalable intrusion detection. In: Proceedings IEEE Information Technology Conference, Syracuse, NY. IEEE Press, New York, pp. 129–133.

Salton, G., 1983. Automated Text Processing. Prentice-Hall, New York.

Stolfo, S.J., Prodromidis, A.L., Tselepis, S., Lee, W., Fan, D., Chan, P.K., 1997. JAM: Java agents for meta-learning over distributed databases. In: Proceedings, KDD97 and AAA197 Workshop on AI Methods in Fraud and Risk Management.

van Doorn, L., 1999. nfsbug.c.. Available from ftp://ftp.cs.vu.nl/pub/leendert/nfsbug.shar.

Warrender, C., Forrest, S., Pealmutter, B., 1999. Detecting intrusions using system calls: alternative data models. In: Proceedings IEEE Symposium on Security and Privacy. IEEE Press, New York.

Yang, J., Honavar, V., 1998. In: Liu, H., Motoda, H. (Eds.), Feature Extraction Construction and Subset Selection: A Data Mining Perspective. Kluwer Academic Publishers, Boston, MA (Chapter feature subset selection using a genetic algorithm).

Yang, J., Pai, P., Honavar, V., Miller, L., 1998. Mobile intelligent agents for document classification and retrieval: A machine learning approach. In: Proceedings of the European Symposium on Cybernetics and Systems Research.

Guy Helmer is a Senior Software Engineer at Palisade Systems, Inc., where he is building network security and applications protocol management appliances. He received his Ph.D. in Computer Science from Iowa State University in 2000, his M.S. in Computer Science from Iowa State University in 1998, and his B.S. in Computer Science from the South Dakota School of Mines and Technology in 1989. Guy spent seven years as a system programmer, network engineer, and system administrator at Dakota state University in Madison, S.D. (one of Yahoo!'s most Wired Universities in 1998, 1999, and 2000), where accomplishments included networking multiple campuses, establishing the first connection for the state government of South Dakota to the Internet, and engineering one of the first dorm networks in the region that connected every dorm room. His research interests include operating system and network security, high-performance computation, software engineering, and software safety.

Johnny Wong is a Full Professor of the Computer Science Department, Iowa State University at Ames, Iowa USA. His research interests include Operating Systems, Distributed Systems, Telecommunication Networks, Broadband-Integrated Services Digital Networks, Concurrency Control and Recovery, Multimedia and Hypermedia Systems, Intelligent Multi-Agents Systems, Intrusion Detection.

He has been an investigator for research contracts with Telecom Australia from 1983 to 1986, studying the performance of network protocols of the ISDN. During this period, he has contributed to the study and evaluation of the communication architecture and protocols of ISDN. From 1989 to 1990, he was the Principal Investigator for a research contract with Microware Systems Corporation at Des Moines, Iowa. This involved the study of Coordinated Multimedia Communication in ISDN. In Summers 1991 and 1992, Dr. Wong was supported by IBM corporation in Rochester. While at IBM, he worked on the Distributed Computing Environment (DCE) for the Application Systems. This involved the integration of communication protocols and distributed database concepts. Dr. Wong is also involved in the Coordinated Multimedia System (COMS) in Courseware Matrix Software Project, funded by NSF Synthesis Coalition Project to enhance engineering education. From 1993 to 1996, he is working on a research project on a knowledge-based system for energy conservation education using multimedia communication technology, funded by the Iowa Energy Center. From 1995 to 1996, he was supported by the Ames Laboratory of the Department of Energy (DOE), working in Middleware for Multidatabases system.

He was involved in projects on Intelligent Multi-Agents for Intrusion Detection and Countermeasures funded by the Department of Defense (DoD), Database Generating and X-Ray Displaying on the World Wide Web Applications funded by Mayo Foundation. Currently, he is working on the CISE Educational Innovation: Integrated Security Curricular Modules and NSF SFS Program on Information Assurance, both funded by the National Science Foundation (NSF).

Vasant Honavar received his Ph.D. in Computer Science and Cognitive Science from the University of Wisconsin, Madison. He directs the Artificial Intelligence Research Laboratory in the Department of Computer Science at Iowa State University where he is currently a full professor. He also serves on the faculties of interdepartmental programs in Information Assurance and Bioinformatics and Computational Biology. His current research and teaching interests include Artificial Intelligence, Machine Learning, Data Mining and Knowledge Discovery, Distributed Learning, Distributed Heterogenous Information Integration, Distributed Information Infrastructures, Intelligent Agents and Multi-Agent Systems, Bioinformatics and Computational Biology, Distributed Artificial Intelligence, Applied Artificial Intelligence, and Information Security. He has published over 100 research papers in refereed journals, books, and conferences and has co-edited three books. He is a Co-Editor-in-Chief of Journal of Cognitive Systems Research and an Associate Editor of Information Sciences Journal. His research has been partially funded by grants from the National Science Foundation, the John Deere Foundation, the Department of Defense, Pioneer Hi-bred, and IBM. Prof. Honavar is a member of ACM, AAAI, IEEE, and the New York Academy of Sciences.

Les Miller is a professor and chair of Computer Science at Iowa State University. His research interests include databases, data warehouses, integration of heterogeneous distributed data sources, and mobile agents. He was the editor of ISMM's International Journal of Microcomputer Applications from 1989 to 1999. He has also served as the vice president of the International Society for Computers and their Applications (ISCA).