The Architecture of a Network Level Intrusion Detection System*

Richard Heady George Luger Arthur Maccabe
Mark Servilla

Department of Computer Science
University of New Mexico
Albuquerque, NM 87131

March 15, 1990

Abstract

This paper presents the preliminary architecture of a network level intrusion detection system. The proposed system will monitor base level information in network packets (source, destination, packet size, and time), learning the 'normal' patterns and announcing anomalies as they occur. The goal of this reserach is to determine the applicability of current intrusion detection technology to the detection of network level intrusions. In particular, we are investigating the possibility of using this technology to detect and react to worm programs.

1 Introduction

Protection of resources is an important aspect of any computing system. Three aspects of network/distributed systems make these systems more vunerable to attack than independent machines: 1) networks typically provide more resources than independent machines, 2) network systems are typically configured to facilitate resource sharing, and 3) global protection policies which are applied to all of the machines in a network are rare.

The research project described in this report is aimed at investigating the applicability of intrusion detection techniques to detect network level intrusions. In particular, we are investigating the possibility of developing a system which can detect and react to worm programs. A "worm" program is characterized by the fact that the program moves from one node in a network to another. The

*This work was supported in part by the Office of Safeguards and Security of the US Dep. of Energy through the Nuclear Safeguards Group (N-4) of Los Alamos National Laboratory.

Internet worm of November 1988 [10] provided ample demonstration of the fact that computer networks are susceptible to this type of attack.

Protection encompasses the *integrity*, *confidentiality*, and *availability* of the resources provided by a computing system. Historically, protection has been provided in the context of a security model [7]. Security models are based on the concept of an *action* which is applied to a set of resources (frequently called *objects*). Each action can be attributed to an individual user, the *initiator* of the action. A security model specifies which actions are permitted based on the initiator of the action, the objects involved in the action, and the context in which the action is requested. Importantly, every action performed in the computing system must be validated by an implementation of the security model.

There are at least three ways in which a computing system based on the security model approach can be compromised: an incorrect implementation of the model, an inaccurate authentication of the user, or an *insider* attack. Any implementation of a security model is at best an approximation of the model. The more complex the model, the more likely it is that there is a discrepancy between the implementation and the model. Any such discrepancy must be viewed as a means by which the integrity, confidentiality, or availability of a resource could be compromised.

The implementation of a security model incorporates an authentication module which is used to identify the individual initiating actions in the system. At best, the authentication module provides a high level of confidence that the individual initiating an action has been correctly identified. Regardless of its complexity, every authentication module can be compromised. When the authentication module is compromised, i.e., an individual is incorrectly identified, the security model no longer provides protection for the resources of the computing system.

Finally, the security model approach does not address the problems associated with an *insider* attack. It is possible that an individual who has been granted the right to manipulate an object may abuse that right. This possibility is not addressed in most security models. As such, a privileged individual can compromise the integrity, confidentiality, or availability of the resources which he or she has been authorized to manipulate.

Given these difficulties, several researchers have proposed that the traditional security model be augmented with an intrusion detection system [6, 9, 8, 12]. Any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource is termed an *intrusion*. An *intruder* is the individual or group of individuals who initiates the actions in the intrusion. Intrusion detection systems are based on the belief that an intrusion will be reflected by a change in the 'normal' patterns of resource usage. As such, intrusion detection systems have been developed to monitor specific types of activities and announce anomalies in the behaviors observed. The anomalies announced by an intrusion detection system serve as an indication that an intrusion may be in progress.

If the intrusion detection system bases its monitoring on the actions performed by an individual (as in the IDES system), the monitoring can be viewed as an on-going authentication process. In this sense, the individual's behavior will continue to authenticate his or her identity as long as those activities are within an acceptable variance of the normal behavior for the individual. However, if the activities performed by an individual are significantly different than the activities normally performed by the individual, there is reason to suspect the individual is not who he or she claims to be – i.e., that an intrusion has occurred.

Like security models, intrusion detection systems are not immune to attack. Because behaviors change over time, intrusion detection systems must be capable of adapting to reflect changes in the actions that they monitor. As such, a careful intruder can 'teach' the intrusion detection system a new behavior pattern which may culminate in invalid access to resources in the system. In this context the intrusion detection system serves to increase the time it takes to compromise the resources of the system and may increase the probability that the intruder will give up or be caught by alternative mechanisms.

The research project described in this report represents an attempt to apply the techniques associated with intrusion detection to the network level of a computing system.

2 Overview

One approach to designing a network security system is to define network behavior patterns that indicate intrusive or improper use of the network and look for the occurrence of those patterns. While such an approach may be capable of detecting known varieties of intrusive behavior, it would allow new or undocumented types of attack to go undetected. As a result, our decision was to build a system which monitors and learns normal network behavior and then detects deviations from it. Our assumption, therefore, is that normal network traffic will be characterized by discernible patterns of data flow, and that intrusive behavior will in some way violate those patterns.

The description of our proposed system design will be divided into two sections:

1. A module which monitors a local network and captures information about data packet transmission. This module will sample the network transmissions to create a statistically valid profile of the full data flow, and it will perform some preprocessing of the saved data.
2. A module which uses the preprocessed information from (1) as input to a classifier system and genetic algorithm which learns normal patterns of network traffic and flags deviations from those patterns.

Figure 1 gives an overview of the system design.

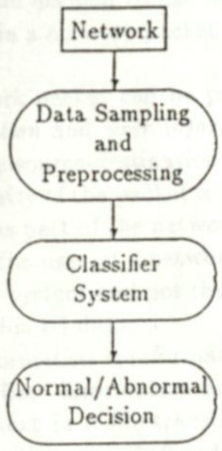

Figure 1: Basic architecture of intrusion detection system.

3 Data Sampling and Preprocessing

The first step in collecting data is to determine exactly what type of data should be collected. Since the goal of this project is directed toward intrusion detection at the network level a natural choice of data is the network transmission packet. The network packet provides us with two types of information to study, transport information and user information. We have chosen to use only the transport information as a primary source of data.

The second step in collecting data is to develop some mechanism for monitoring network packets. Since detecting an intrusion is not dependent on the specific method used to monitor packets, any mechanism capable of obtaining a valid data sampling is satisfactory. Currently, we are using a software package that allows monitoring of an Ethernet network. To our disadvantage, this particular software does not provide a valid sampling of network packets due to a significant loss of packets during the monitoring phase. Although we cannot base any results on the information provided by this software, we can use the information to observe the classifier system.

The final step in collecting data is to process it in such a way that it is compressed in size and in a format acceptable to the classifier system. In practice, this preprocessing phase is very simple to implement.

3.1 A Choice of Packet Data

In the case of network traffic we decided to use only unbiased data. Unbiased data is simply the information in a network packet that cannot be made deceptive by a fraudulent user.

In practical terms, a network packet can be partitioned into two forms of information, *transport information* and *user information*[10]. Transport information generally consists of the source-destination ordered pair and some type of checksum on which the integrity of the packet is determined. Transport information is added to the packet as part of the network transmission protocol and cannot be directly affected by the user of a network. In other words, transport information is an artifact of the system and not the user. We therefore consider transport information to be unbiased data.

On the other hand, user information is information which the user wishes to transport across the network. User information may vary from individual key strokes to large aggregates of text in a file transfer. This type of information can be directly manipulated by the user. A fraudulent user can easily modify this information to be deceptive, and we therefore consider user information to be biased data.

3.2 The Physical Connection

Presently, all data collection takes place on a SUN Microsystems 3/60 workstation provided by the Computer and Information Resources and Technology (CIRT) center at the University of New Mexico (UNM). The SUN 3/60 is connected to an intra-center Ethernet which receives external network traffic via the Campus Data Communication Network (CDCN). The CDCN is a broadband network and is the backbone along which UNM traffic is handled.

To monitor Ethernet traffic, we use the Network Interface Tap (NIT) facility provided by SUN Microsystems as part of their SUN Operating System network software utilities[1]. At this time, NIT is the only software available on our hardware configurations which allows promiscuous access to Ethernet traffic. There is, however, a serious problem involving continuous data collection when using NIT. A significant loss of Ethernet packets are attributed to monitoring on a multitasking system and to an unknown error in the NIT software. Both problems will be discussed in greater detail below. Let us first provide an overview of NIT.

3.2.1 Network Interface Tap (NIT)

NIT is a facility composed of several *streams* modules and drivers which provide link-level network access. As such, NIT is capable of both reading from and writing to the Ethernet device. NIT performs this service by placing itself between the Ethernet device and a user process. When NIT is initialized as a reading device, it attempts to copy the packets which enter the Ethernet device buffer and return them as a stream of data. When initialized as a writing device, NIT requires the user process to supply an input stream of data which is then transmitted out onto the network through the Ethernet device. The components which collectively provide this service are the *interface* (nit_if), *packet filter* (nit_pf), and *buffering* (nit_buf) modules.

The Interface Module The primary component of NIT is the interface module[2]. The interface module is a *streams* device driver which interacts directly with the system's Ethernet device. The interface module transcribes packets from the Ethernet device to the read side of the stream or from the write side of the stream to the Ethernet device for transmission on the network. When NIT is opened as a reading device, the interface module provides additional information which may be prepended to the transcribed packet if desired. The information includes the size of the received packet, a timestamp marking the time of reception, and a cumulative count of dropped packets from the time the device was first opened[1].

The Packet Filter Module An optional module provided by NIT is the packet filter module[3]. The packet filter module operates only when NIT is opened as a reading device. The module subjects each packet to a filter which passes only those packets that the filter accepts on to its upstream destination.

The Buffering Module Also optional, the buffering module can be used to increase system efficiency[4]. The buffering module places an internal buffer between the Ethernet device and the user process. Packets which are copied by NIT are buffered into larger aggregates, thereby reducing the overhead incurred by repeated reads of the Ethernet device buffer.

3.2.2 Problems Inherent to NIT

A serious problem that we have encountered with the NIT facility is significant packet loss during the monitoring phase. At the present time, we can only hypothesize as to the cause of this loss. As described above, NIT provides intermediate buffering between the Ethernet device and the stream device at the user level. When operating in promiscuous mode, NIT attempts to copy all of the packets which are processed by the Ethernet device into its own buffer. But, NIT is also a process running in a multitasking environment. Inevitably, NIT is pre-empted by the operating system scheduler in favor of another process. While NIT is in a pre-empted state the Ethernet device continues to process

[1]NIT software documentation explains that dropped packets are a result of system constraints. Presumably, these "system constraints" are one possible source of our sampling problems.

incoming packets; these packets are never seen by NIT. The ensuing problem results in a packet loss of approximately 75 percent. The percentage of loss was empirically calculated by recording the cumulative sizes of identifiable packets during a file transfer over the network and comparing that value with the actual size of the file. Observed losses were in the range of 75 to 95 percent.

Another problem which we have associated with the NIT facility is the possibility that internal data structures are being destroyed during periods of execution longer than 20 seconds. As a result, the system is left in an indeterminate state and must be rebooted to recover. To remedy this problem, the NIT interface is reset at 5 seconds intervals by the monitor application program. Resetting the interface during the monitoring phase incurs additional packet losses.

Realizing that the percentage of packet loss is too great for valid testing purposes, we are now pursuing alternative methods for data collection.

3.2.3 The Monitor Application Program

The network monitor application is comprised of the NIT interface module and the NIT buffering module. We find no practical use at this time to filter any of the incoming packets by using the NIT packet filter module. Functionally, the monitor program polls the read side of the NIT stream device for a specified length of time collecting packets.

Prior to reading the stream device NIT is configured in the following order. First, the stream device is opened with the C system call *open*. After opening the NIT stream device, the buffering module is configured and the NIT device is bound to the SUN Ethernet device interface. Finally, the read buffer is flushed to remove anything that may have accumulated before the device reached its final configuration. Once the configuration is accomplished, the monitoring program executes a polling loop which continues until a specified time-out occurs. Within the polling loop two events take place: packet information is read from the NIT stream device and written to a file, and the NIT interface is reset at 5 second intervals[2]. Resetting the NIT interface is required to prevent the indeterminate state problem described above. The relationship between the Ethernet device, the NIT facility, and the monitor application program is demonstrated in Figure 2. The reader is referred to the Appendix for a source listing of the monitor program.

As mentioned earlier, only the transport information portion of the network packet is recorded. All other information is discarded by either the NIT buffering module or the monitor application program. The NIT buffering module is configured to build aggregates of only the first 58 bytes of the Ethernet packet. The 58 bytes include the prepended packet information which the NIT interface module supplies, the Ethernet packet header information, and the Internet packet header information if the network transmission protocol is TCP/IP. Once

[2]Resetting the NIT interface takes place only if the time-out value is greater than 5 seconds.

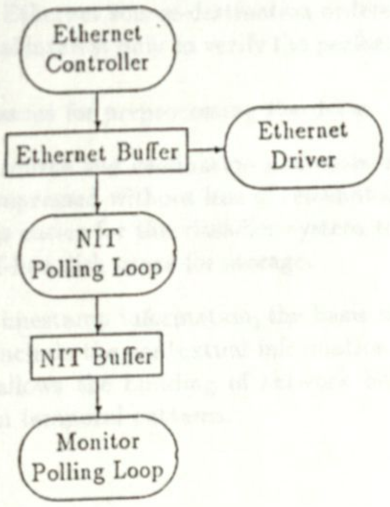

Figure 2: NIT interface facility.

the aggregate is available from the NIT stream device, the monitor application program 1) reads the aggregate into a temporary buffer, 2) filters out the packet length value, the cumulative packet drop value[3], the timestamp value, and the Ethernet source-destination ordered pair from the buffer, and 3) writes this information to a file. A total of 28 bytes from each observed packet is recorded.

During an average monitoring session we are able to collect approximately 160 packets in a one second interval. This results in a file growth rate of 4,480 bytes per second or more than 1 Megabyte every four minutes. Remembering that NIT loses a large number of packets, this growth rate reflects only 25 percent of the actual Ethernet traffic on our network. Data accumulation at this rate will inundate even the largest storage disks after a day of monitoring the network. Even with data compression, we realize that the amount of data collected during a 24 hour monitoring session would be overwhelming. Therefore, we are currently looking at various methods of discrete sampling to reduce the amount of data to be collected.

3.3 Data Preprocessing

Of the data fields currently saved, there are only four of the five types which are important to the classifier system. These are the packet size value, the times-

[3] To date, we have been unable to determine the meaning of the number in the packet drop field.

tamp value, and the Ethernet source-destination ordered pair[4]. The cumulative packet drop value is of interest only to verify the performance of our monitoring application program.

There are two reasons for preprocessing the data:

1. In the cases of source and destination addresses and packet sizes, the raw data can be compressed without loss of relevant information. This results in data which is easier for the classifier system to manipulate and which requires less off-line disk space for storage.
2. In the case of timestamp information, the basic second count provided is augmented to include the contextual information of hour of day and day of week. This allows the building of network behavior profiles that are based on human temporal patterns.

3.3.1 Address

Representation of an Ethernet address, whether destination or source, requires 6 bytes. This ensures that there will be a sufficient number of distinct addresses available for each unique Ethernet controller within a computer system. However, we will only see a very small set of these addresses in any given LAN. Therefore, each Ethernet address in our LAN will be mapped into a two byte value.

3.3.2 Size

The size of an Ethernet packet can vary from 64 to 1518 bytes[10]. However, the packets can be readily grouped into a much smaller number of categories. For example, a single keystroke packet may contain transport information and the data which represents the keystroke – one or two bytes. In contrast, a file transfer program will attempt to utilize the largest available packet size possible to transfer a large aggregate of data in a single packet. Data size information, therefore, can be compressed into groups of naturally occurring clusters. We expect the reduction to be from 1518 to on the order of 4 to 16 categories.

3.3.3 Time

The timestamp value provided by the NIT facility is a relative time based on the absolute time 00:00:00 GMT January 1, 1970. To capture the full potential of time information we will transform the NIT timestamp to one that includes second, minute, hour, and day of week information.

[4]We refer to the Ethernet source-destination ordered pair as if it is a single value. It is actually comprised of two separate and independent values.

4 Learning Normal Network Patterns and Flagging Deviations: The Classifier System and Genetic Algorithm

The input for this module of the system consists of simple packet transmission 4-tuples. These 4-tuples include the packet's source, destination, size, and time of transmission, with the time including the contextual information of hour of day and day of week. While these individual events are simple enough, over time a great deal can be inferred from them about the operation of a network. They can be grouped in an impressive number of ways. A brief list of possible characteristics that might be relevant to the problem at hand includes:

- Has a packet transmission been received from a previously uncatalogued source?
- Is the total network traffic over the last n minutes within the expected bounds for this hour of day and day of the week?
- Is the pattern of transmission from a given source to all other processors in the network over the last hour outside the bounds of expected difference by some statistical measure for this time of the day?
- Does the shape of a size histogram of all packet transmissions over the last n minutes match the normal shape of such a histogram closely enough? How about for a given source-destination pair, or for some subset of all source-destination pairs?
- Has the number of packet transmissions increased every minute for the last n minutes?
- Has some combination of the above events occurred in the last n minutes?

The list of questions and categories can be enlarged indefinitely. The point is that individual events acquire meaning only by being given a context. The context that is needed is a set of categories, existing in time, into which individual events and parts of events can be placed. The analysis of these categories can then reveal patterns of normal network behavior, which can in turn be used to detect abnormal behavior.

Proper choice of meaningful categories, however, is a difficult task. The difficulty lies in the fact that patterns of normal network behavior will not be apparent unless good classification of event categories are chosen. A good category, however, is by definition one which reveals patterns of normal behavior.

For instance, suppose the number of packet transmissions between a source-destination pair over a ten minute period is chosen as a category to be monitored. Whether this is indeed a meaningful category will depend on whether a statistical characterization of the source-destination transmissions over ten minutes can be found such that normal network behavior falls within the characterization and intrusive or questionable behavior falls outside of it. This is a question that can only be answered by experience. We are left with a circular definition, which suggests an iterative approach to defining categories.

The design we plan to implement is based on the classifier system and genetic algorithm model described by Booker, Goldberg, and Holland [5]. The model consists of three main components:

1. A classifier system, which is a parallel, message-passing, rule-based system. The system uses rules that are sensitive to input messages, as well as rules that integrate messages from other rules, to eventually develop an output message.

2. A credit assignment algorithm, which evaluates the usefulness and predictive power of individual rules. Each rule has a strength factor which is increased or decreased each time a rule is part of a chain that effects the output message, depending on whether the rule contributed to a correct or incorrect prediction.

3. A genetic algorithm that periodically modifies the set of rules in the classifier system. Rules whose low strength factors indicate they are poor predictors are removed. They are replaced by new rules formed by combining parts of rules whose high strength factors indicate they are good predictors.

The components of the model, illustrated in Figure 3 are now described in more detail.

4.1 The Classifier System

As stated above, the classifier system is a parallel, message-passing, rule-based system. All rules are of the condition/action form; the condition is the receipt of messages containing information that activates the rule, and the action is the sending of messages when the rule is satisfied. All messages contain a tag specifying their origin and an information field. For concreteness, all messages can be thought of as being a bit string of fixed length. The first n bits could represent the tag field. Each of the remaining bits could encode the presence or absence of some simple condition, depending on whether they contained a 0 or a 1.

A classifier system, then, consists of four parts:

1. An input interface. In our case this will be a message that contains the information from a 4-tuple describing an individual packet transmission.

2. The classifiers. These are rules which define the ways in which the system consumes and creates messages.

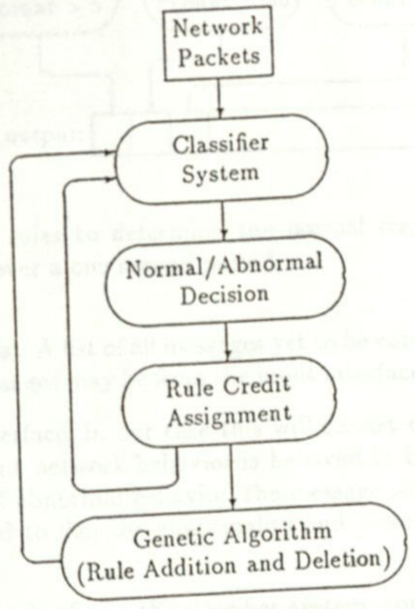

Figure 3: The classifier system learning and decision process, with feedback from the credit assignment and genetic algorithms.

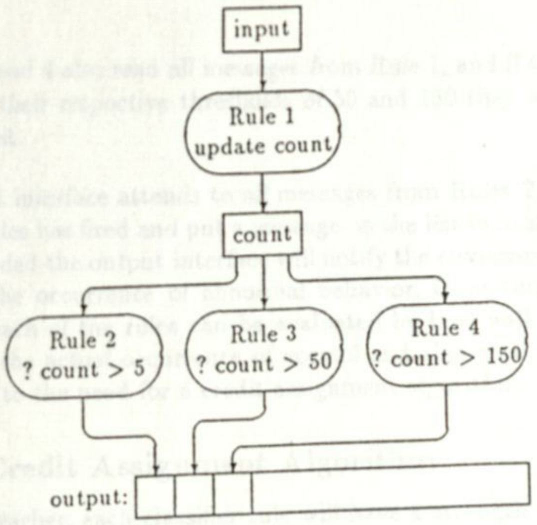

Figure 4: Classifier rules to determine the normal transmission threshold of packets of size 100 over a one second period.

3. The message list. A list of all messages yet to be considered by the classifier rules. The messages may be from the input interface or from satisfied rules.

4. An output interface. In our case this will consist of a message indicating whether current network behavior is believed to be normal or abnormal. In the event of abnormal behavior, the message may also contain a list of rules that fired to flag the abnormality and a confidence factor for each rule.

As a simple example of how the classifier system works, suppose that transmissions of packets of size 100 were being considered as an indicator of normal network behavior. Suppose also that we were interested in the number of packets of size 100 over a one second period and that we wished to evaluate 5, 50, and 150 as possible thresholds of abnormality. Then we would have the following four classifier rules, illustrated in Figure 4.

- Rule 1 would read all messages from the input interface. It would use the size and time values in those messages to maintain a count of packets of size 100 over a sliding time window of one second. After processing an input message it will put a message of its own on the message list with the updated count for the last second.

- Rule 2 reads all messages put on the message list by Rule 1. If the current count of packets of size 100 over the last second exceeds 5 then Rule 2 in turn puts a message on the message list saying its threshold has been crossed.

- Rules 3 and 4 also read all messages from Rule 1, and if the current count exceeds their respective thresholds of 50 and 150 they also put messages on the list.

The output interface attends to all messages from Rules 2, 3, and 4. When any of those rules has fired and put a message on the list indicating its threshold has been exceeded the output interface will notify the environment that the rule is predicting the occurrence of abnormal behavior. Over time the thresholds embedded in each of the rules can be evaluated by how well their predictions correlate with the actual occurrence of normal and abnormal network activity. This brings us to the need for a credit assignment algorithm.

4.2 The Credit Assignment Algorithm

As mentioned earlier, each classifier rule will have a strength factor associated with it. When it is created a rule's strength factor will be initialized to some standard value. Each time a rule fires and puts a message on the message list its strength factor will be decremented. If feedback through the output interface shows that the rule was correct in its prediction of the type of activity that was occurring then it will be rewarded by having its strength factor increased. In the example just described, if it was in fact perfectly normal to see 120 transmissions of packets of size 100 over a one second period then the rules with thresholds of 5 and 50 would fire frequently. Each time they posted a message they would pay for the privilege with part of their strength factor. If they were never rewarded for having correctly predicted abnormal activity, their strength would gradually dissipate and they would become candidates for removal by the genetic algorithm. The rule with threshold of 150, however, might only fire when abnormal activity was in fact occurring. In that case its strength factor would increase. That increase would ensure its continued existence and also make it a candidate for combination with other strong rules under the genetic algorithm to make new conjunctive rules.

Unfortunately, the assignment of credit to rules is not quite so straightforward as the discussion so far might indicate. For instance, consider the classifier rules represented in Figure 5. Rules 1 and 2 both read the input message and fire when they detect the presence of some condition. Rule 3 reads messages from both 1 and 2 and fires only when both of them have fired. Messages from all three rules are read by the output interface. It may be that Rules 1 and 2 predict abnormality too often individually, but that in conjunction, as Rule 3, they are very accurate predictors. If reenforcement for correct prediction only came directly from the output interface, then 1 and 2 would grow weaker over time even though they had considerable value when used together in 3.

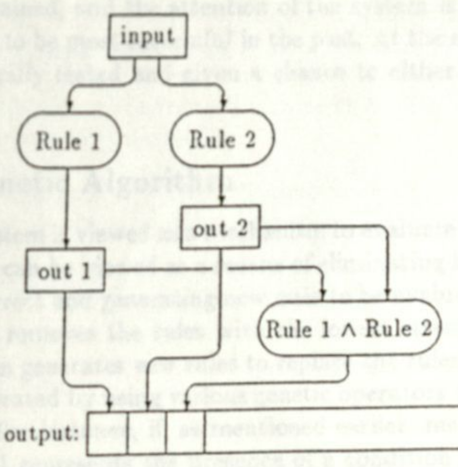

Figure 5: An example illustrating the need for a "bucket brigade" credit assignment algorithm.

The solution to the problem is in what is known as a "bucket brigade" algorithm. The algorithm is based on the recognition that many rules which send messages to the output interface are actually the result of longer, more complex reasoning chains. When a rule is given credit by the output interface for a correct prediction, therefore, it in turn passes some of that credit back up to all of the rules that had to fire in order for it to fire. These contributing rules, in their turn, pass back some portion of the credit they receive to the rules that enabled them to fire. The result is that all rules in the chain that leads to the rule that ultimately makes a prediction receive some of the reward when that prediction is correct. Returning to the simple example of the preceding paragraph, when Rule 3 is correct it takes some fraction of the increase to its own strength factor, divides it evenly, and passes the resulting strength increases back to Rules 1 and 2.

In addition to evaluating rules to guide their manipulation by the genetic algorithm, strength factors can also be used to manage the size of the computation performed by the classifier system after each input message. The classifier system is essentially a system to generate and test hypotheses. Especially in its early stages it may consist of a very large number of rules. If the number of messages generated by each event becomes larger than desired, then the rules governing the firing of a given classifier rule can be changed. Rather than firing every time it receives all the messages that enable it to fire, a rule only fires probabilistically, based on its strength factor. That is, the higher its strength factor, the higher the chance it will fire when its conditions are met. The lower its strength factor, the lower the chance it will fire. The size of each computation is thereby constrained, and the attention of the system is focused on the rules that have proven to be most successful in the past. At the same time, even weak rules are periodically tested and given a chance to either increase or decrease their strength.

4.3 The Genetic Algorithm

If the classifier system is viewed as a mechanism to evaluate hypotheses, then the genetic algorithm can be viewed as a means of eliminating hypotheses that have proven to be incorrect and generating new ones to be evaluated. Periodically the genetic algorithm removes the rules with the lowest strength factors from the list of rules. It then generates new rules to replace the rules it has removed. The new rules are generated by using various genetic operators to join together parts of existing rules. For instance, if, as mentioned earlier, messages are viewed as bit strings where 1 represents the presence of a condition and 0 represents the absence of a condition, then new rules can be created by switching the conditions required over some substring of the messages between two rules.

Rather than simply using the rules with the highest strength factors to generate the new rules, strength factors are used to generate a probability that a given rule will be used in new rule generation. Just using the strongest rules would run the danger of biasing the search of the possible rule space too heavily based on the order in which events in the event space were encountered. Occasionally generating new rules using some of the weaker existing rules allows exploration of parts of the rule space that might otherwise be completely ignored.

4.4 The Feedback Loop Problem in a Network Security System

The classifier system requires feedback from the environment in order to assign credit to its individual rules. In the context of computer network security that means that a decision must be made whether the current state of data flow in the network is normal or abnormal. Without knowledge about the current state the predictions of normality or abnormality made by individual rules cannot be determined to be either correct or incorrect.

Initially the classifier system can be considered to be in a learning phase, and all encountered network behavior can be considered normal. At some point, however, it will be necessary to present the system with examples of actual abnormal conditions in order to make sure that these are not simply regarded as normal as well. The problem, of course, is that very little information is available about the appearance of a security attack at the network level.

We currently plan to investigate the following approaches:

1. Obtain records of actual network attacks, simulate them, and test whether the security system detects them.

2. Generate arbitrary situations of abnormal network data flow and test whether the system detects them.

3. Generate our own scenarios of network intrusion or attack, implement and execute them, and test whether the system detects them.

4. Monitor an operating network and wait for an actual intrusion to occur.

5 Summary

Considerable work has been done in the area of computer system protection. For the most part, however, that work has concentrated on protection at the level of individual machines and operating systems. The widespread existence of computer networks, combined with events such as the Internet worm of November, 1988, demonstrate the need to address protection issues at the network level as well. The focus of our research is to determine the feasibility of network level monitoring to protect network resources from attack.

Our initial goal is to build an off-line prototype system capable of learning normal patterns of network use and flagging departures from those patterns of normality. Such a system will permit verification of the hypothesis that intrusive attacks are in fact detectable as deviations from a rule-based profile of normal behavior. In addition, the prototype will allow us to establish whether the behavior profile eventually reaches a state of statistical stability. Our belief is that after an initial learning phase during which both rules and the confidence factors attached to them vary rapidly a state of equilibrium will be reached. At that point the system will need to be flexible enough to adjust to genuine incremental changes in normal network behavior, but not so flexible as to allow an intruder to gradually teach the system to accept new behavior which opens the network to attack.

Our longer term research goals include moving from an off-line to an on-line system in order to provide real-time network level protection. The move to an on-line system will in turn raise the issue of developing appropriate reactions to detected intrusions. Attempts to lessen the impact of detected intrusions may include delaying or ignoring communications involving the suspected participating nodes.

References

[1] *Sun Microsystem's Operating System* 4.0. Section nit (4P): Protocols.

[2] *Sun Microsystem's Operating System* 4.0. Section nit_if (4M): Devices and Network Interfaces.

[3] *Sun Microsystem's Operating System* 4.0. Section nit_pf (4M): Devices and Network Interfaces.

[4] *Sun Microsystem's Operating System* 4.0. Section nit_buf (4M): Devices and Network Interfaces.

[5] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, September 1989.

[6] Dorothy E. Denning. An intrusion-detection model. In *IEEE Symposium on Security and Privacy*, pages 118–131. IEEE, 1986.

[7] Carl E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[8] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *IEEE Symposium on Security and Privacy*, pages 59–66. IEEE, 1988.

[9] T.F. Lunt, R. Jagannathan, R. Lee, S. Listgarten, D.L. Edwards, P.G. Neumann, H.S. Javitz, and A. Valses. Ides: The enhanced prototype. Technical report, SRI International, October 1988.

[10] Eugene F. Spafford. The internet worm: Crisis and aftermath. *Communications of the ACM*, 32(6):678–687, June 1989.

[11] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, Englewood Cliffs, N. J., second edition, 1988.

[12] H. S. Vaccaro and G. E. Liepins. Detection of anomalous copmuter session activity. In *IEEE Symposium on Security and Privacy*, pages 280–289. IEEE, 1989.